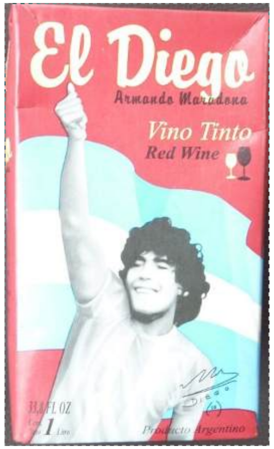
El Diego
Armando Maradona
Vino Tinto
Red Wine
Producto Argentino

Índice

1. algorithm

#include <algorithm> #include <numeric>

Algo	**Params**	**Funcion**
sort, stable_sort	f, l	ordena el intervalo
nth_element	f, nth, l	*void* ordena el n-esimo, y particiona el resto
fill, fill_n	f, l / n, elem	*void* llena [f, l) o [f, f+n) con elem
lower_bound, upper_bound	f, l, elem	*it* al primer / ultimo donde se puede insertar elem para que quede ordenada
binary_search	f, l, elem	*bool* esta elem en [f, l)
copy	f, l, resul	hace resul+i=f+i $\forall i$
find, find_if, find_first_of	f, l, elem / pred / f2, l2	*it* encuentra i $\in$[f,l) tq. i=elem, pred(i), i$\in$[f2,l2)
count, count_if	f, l, elem/pred	cuenta elem, pred(i)
search	f, l, f2, l2	busca [f2,l2) $\in$ [f,l)
replace, replace_if	f, l, old / pred, new	cambia old / pred(i) por new
reverse	f, l	da vuelta
partition, stable_partition	f, l, pred	pred(i) ad, !pred(i) atras
min_element, max_element	f, l, [comp]	*it* min, max de [f,l]
lexicographical_compare	f1,l1,f2,l2	*bool* con [f1,l1]¡[f2,l2]
next/prev_permutation	f,l	deja en [f,l) la perm sig, ant
set_intersection, set_difference, set_union, set_symmetric_difference,	f1, l1, f2, l2, res	[res, ...) la op. de conj
push_heap, pop_heap, make_heap	f, l, e / e /	mete/saca e en heap [f,l), hace un heap de [f,l)
is_heap	f,l	*bool* es [f,l) un heap
accumulate	f,l,i,[op]	$T = \sum$/oper de [f,l)
inner_product	f1, l1, f2, i	T = i + [f1, l1) . [f2, ...)
partial_sum	f, l, r, [op]	r+i = $\sum$/oper de [f,f+i] $\forall i$ $\in$[f,l)
__builtin_ffs	unsigned int	Pos. del primer 1 desde la derecha
__builtin_clz	unsigned int	Cant. de ceros desde la izquierda.
__builtin_ctz	unsigned int	Cant. de ceros desde la derecha.
__builtin_popcount	unsigned int	Cant. de 1's en x.
__builtin_parity	unsigned int	1 si x es par, 0 si es impar.
__builtin_XXXXXXll	unsigned ll	= pero para long long's.

2. Estructuras

2.1. RMQ (static)

Dado un arreglo y una operacion asociativa *idempotente*, get(i, j) opera sobre el rango [i, j). Restriccion: LVL ≥ ceil(logn); Usar [ ] para llenar arreglo y luego build().

```
struct RMQ{
  #define LVL 10
  tipo vec[LVL][1<<(LVL+1)];
  tipo &operator[](int p){return vec[0][p];}
  tipo get(int i, int j) {//intervalo [i,j)
    int p = 31-__builtin_clz(j-i);
    return min(vec[p][i],vec[p][j-(1<<p)]);
  }
  void build(int n) {//O(nlogn)
    int mp = 31-__builtin_clz(n);
    forn(p, mp) forn(x, n-(1<<p))
      vec[p+1][x] = min(vec[p][x], vec[p][x+(1<<p)]);
  }};
```

2.2. RMQ (dynamic)

```
//Dado un arreglo y una operacion asociativa con neutro, get(i, j) opera sobre el rango [i, j).
#define MAXN 100000
#define operacion(x, y) max(x, y)
const int neutro=0;
struct RMQ{
  int sz;
  tipo t[4*MAXN];
  tipo &operator[](int p){return t[sz+p];}
  void init(int n){//O(nlgn)
    sz = 1 << (32-__builtin_clz(n));
    forn(i, 2*sz) t[i]=neutro;
  }
  void updall(){//O(n)
    dforn(i, sz) t[i]=operacion(t[2*i], t[2*i+1]);}
  tipo get(int i, int j){return get(i,j,1,0,sz);}
  tipo get(int i, int j, int n, int a, int b){//O(lgn)
    if(j<=a || i>=b) return neutro;
    if(i<=a && b<=j) return t[n];
    int c=(a+b)/2;
    return operacion(get(i, j, 2*n, a, c), get(i, j, 2*n+1, c, b));
  }
  void set(int p, tipo val){//O(lgn)
    for(p+=sz; p>0 && t[p]!=val;){
      t[p]=val;
      p/=2;
      val=operacion(t[p*2], t[p*2+1]);
    }
  }
}rmq;
//Usage:
cin >> n; rmq.init(n); forn(i, n) cin >> rmq[i]; rmq.updall();
```

2.3. RMQ (lazy)

```
//Dado un arreglo y una operacion asociativa con neutro, get(i, j) opera sobre el rango [i, j).
typedef int Elem;//Elem de los elementos del arreglo
typedef int Alt;//Elem de la alteracion
#define operacion(x,y) x+y
const Elem neutro=0; const Alt neutro2=0;
#define MAXN 100000
struct RMQ{
  int sz;
  Elem t[4*MAXN];
  Alt dirty[4*MAXN];//las alteraciones pueden ser de distinto Elem
  Elem &operator[](int p){return t[sz+p];}
  void init(int n){//O(nlgn)
    sz = 1 << (32-__builtin_clz(n));
    forn(i, 2*sz) t[i]=neutro;
    forn(i, 2*sz) dirty[i]=neutro2;
  }
  void push(int n, int a, int b){//propaga el dirty a sus hijos
    if(dirty[n]!=0){
      t[n]+=dirty[n]*(b-a);//altera el nodo
      if(n<sz){
        dirty[2*n]+=dirty[n];
        dirty[2*n+1]+=dirty[n];
      }
      dirty[n]=0;
    }
  }
  Elem get(int i, int j, int n, int a, int b){//O(lgn)
```

```
    if(j<=a || i>=b) return neutro;
    push(n, a, b);//corrige el valor antes de usarlo
    if(i<=a && b<=j) return t[n];
    int c=(a+b)/2;
    return operacion(get(i, j, 2*n, a, c), get(i, j, 2*n+1, c, b));
  }
  Elem get(int i, int j){return get(i,j,1,0,sz);}
  //altera los valores en [i, j) con una alteracion de val
  void alterar(Alt val, int i, int j, int n, int a, int b){//O(lgn)
    push(n, a, b);
    if(j<=a || i>=b) return;
    if(i<=a && b<=j){
      dirty[n]+=val;
      push(n, a, b);
      return;
    }
    int c=(a+b)/2;
    alterar(val, i, j, 2*n, a, c), alterar(val, i, j, 2*n+1, c, b);
    t[n]=operacion(t[2*n], t[2*n+1]);//por esto es el push de arriba
  }
  void alterar(Alt val, int i, int j){alterar(val,i,j,1,0,sz);}
}rmq;
```

2.4. RMQ (persistente)

```
typedef int tipo;
tipo oper(const tipo &a, const tipo &b){
    return a+b;
}
struct node{
  tipo v; node *l,*r;
  node(tipo v):v(v), l(NULL), r(NULL) {}
    node(node *l, node *r) : l(l), r(r){
        if(!l) v=r->v;
        else if(!r) v=l->v;
        else v=oper(l->v, r->v);
    }
};
node *build (tipo *a, int tl, int tr) {//modificar para que tome tipo a
  if (tl+1==tr) return new node(a[tl]);
  int tm=(tl + tr)>>1;
  return new node(build(a, tl, tm), build(a, tm, tr));
}
node *update(int pos, int new_val, node *t, int tl, int tr){
  if (tl+1==tr) return new node(new_val);
  int tm=(tl+tr)>>1;
  if(pos < tm) return new node(update(pos, new_val, t->l, tl, tm), t->r);
  else return new node(t->l, update(pos, new_val, t->r, tm, tr));
}
tipo get(int l, int r, node *t, int tl, int tr){
    if(l==tl && tr==r) return t->v;
  int tm=(tl + tr)>>1;
    if(r<=tm) return get(l, r, t->l, tl, tm);
    else if(l>=tm) return get(l, r, t->r, tm, tr);
  return oper(get(l, tm, t->l, tl, tm), get(tm, r, t->r, tm, tr));
}
```

2.5. Fenwick Tree

```
//For 2D threat each column as a Fenwick tree, by adding a nested for in each operation
struct Fenwick{
  static const int sz=1000001;
  tipo t[sz];
  void adjust(int p, tipo v){//valid with p in [1, sz), O(lgn)
    for(; p<sz; p+=(p&-p)) t[p]+=v; }
  tipo sum(int p){//cumulative sum in [1, p], O(lgn)
    tipo s=0;
    for(; p; p-=(p&-p)) s+=t[p];
    return s;
  }
  tipo sum(int a, int b){return sum(b)-sum(a-1);}
  //get largest value with cumulative sum less than or equal to x;
  //for smallest, pass x-1 and add 1 to result
  int getind(tipo x) {//O(lgn)
      int idx = 0, mask = N;
      while(mask && idx < N) {
        int t = idx + mask;
      if(x >= tree[t])
          idx = t, x -= tree[t];
        mask >>= 1;
      }
      return idx;
  }};
```

2.6. Union Find

```
struct UnionFind{
  vector<int> f;//the array contains the parent of each node
  void init(int n){f.clear(); f.insert(f.begin(), n, -1);}
  int comp(int x){return (f[x]==-1?x:f[x]=comp(f[x]));}//O(1)
  bool join(int i, int j) {
    bool con=comp(i)==comp(j);
    if(!con) f[comp(i)] = comp(j);
    return con;
  }};
```

2.7. Disjoint Intervals

```
bool operator< (const ii &a, const ii &b) {return a.fst<b.fst;}
//Stores intervals as [first, second]
//in case of a collision it joins them in a single interval
struct disjoint_intervals {
  set<ii> segs;
  void insert(ii v) {//O(lgn)
    if(v.snd-v.fst==0.) return;//OJO
    set<ii>::iterator it,at;
    at = it = segs.lower_bound(v);
    if (at!=segs.begin() && (--at)->snd >= v.fst)
      v.fst = at->fst, --it;
    for(; it!=segs.end() && it->fst <= v.snd; segs.erase(it++))
      v.snd=max(v.snd, it->snd);
    segs.insert(v);
  }
};
```

2.8. RMQ (2D)

```
struct RMQ2D{
  static const int sz=1024;
  RMQ t[sz];
  RMQ &operator[](int p){return t[sz/2+p];}
  void build(int n, int m){//O(nm)
    forr(y, sz/2, sz/2+m)
      t[y].build(m);
    forr(y, sz/2+m, sz)
      forn(x, sz)
        t[y].t[x]=0;
    dforn(y, sz/2)
      forn(x, sz)
        t[y].t[x]=max(t[y*2].t[x], t[y*2+1].t[x]);
  }
  void set(int x, int y, tipo v){//O(lgm.lgn)
    y+=sz/2;
    t[y].set(x, v);
    while(y/=2)
      t[y].set(x, max(t[y*2][x], t[y*2+1][x]));
  }
  //O(lgm.lgn)
  int get(int x1, int y1, int x2, int y2, int n=1, int a=0, int b=sz/2){
    if(y2<=a || y1>=b) return 0;
    if(y1<=a && b<=y2) return t[n].get(x1, x2);
    int c=(a+b)/2;
    return max(get(x1, y1, x2, y2, 2*n, a, c),
         get(x1, y1, x2, y2, 2*n+1, c, b));
  }
};
//Example to initialize a grid of M rows and N columns:
RMQ2D rmq;
forn(i, M)
  forn(j, N)
    cin >> rmq[i][j];
rmq.build(N, M);
```

2.9. Big Int

```
#define BASEXP 6
#define BASE 1000000
#define LMAX 1000
struct bint{
    int l;
    ll n[LMAX];
    bint(ll x=0){
        l=1;
        forn(i, LMAX){
            if (x) l=i+1;
            n[i]=x%BASE;
            x/=BASE;

        }
    }
    bint(string x){
```

```
  l=(x.size()-1)/BASEXP+1;
    fill(n, n+LMAX, 0);
    ll r=1;
    forn(i, sz(x)){
      n[i / BASEXP] += r * (x[x.size()-1-i]-'0');
      r*=10; if(r==BASE)r=1;
    }
  }
  void out(){
  cout << n[l-1];
  dforn(i, l-1) printf("%6.6llu", n[i]);//6=BASEXP!
 }
 void invar(){
  fill(n+l, n+LMAX, 0);
  while(l>1 && !n[l-1]) l--;
 }
};
bint operator+(const bint&a, const bint&b){
 bint c;
  c.l = max(a.l, b.l);
  ll q = 0;
  forn(i, c.l) q += a.n[i]+b.n[i], c.n[i]=q %BASE, q/=BASE;
  if(q) c.n[c.l++] = q;
  c.invar();
  return c;
}
pair<bint, bool> lresta(const bint& a, const bint& b)   // c = a - b
{
 bint c;
  c.l = max(a.l, b.l);
  ll q = 0;
  forn(i, c.l) q += a.n[i]-b.n[i], c.n[i]=(q+BASE) %BASE, q=(q+BASE)/BASE-1;
  c.invar();
  return make_pair(c, !q);
}
bint& operator-= (bint& a, const bint& b){return a=lresta(a, b).first;}
bint operator- (const bint&a, const bint&b){return lresta(a, b).first;}
bool operator< (const bint&a, const bint&b){return !lresta(a, b).second;}
bool operator<= (const bint&a, const bint&b){return lresta(b, a).second;}
bool operator==(const bint&a, const bint&b){return a <= b && b <= a;}
bint operator*(const bint&a, ll b){
  bint c;
  ll q = 0;
  forn(i, a.l) q += a.n[i]*b, c.n[i] = q %BASE, q/=BASE;
  c.l = a.l;
  while(q) c.n[c.l++] = q %BASE, q/=BASE;
  c.invar();
  return c;
}
bint operator*(const bint&a, const bint&b){
  bint c;
  c.l = a.l+b.l;
  fill(c.n, c.n+b.l, 0);
  forn(i, a.l){
    ll q = 0;
    forn(j, b.l) q += a.n[i]*b.n[j]+c.n[i+j], c.n[i+j] = q %BASE, q/=BASE;
    c.n[i+b.l] = q;
  }
  c.invar();
  return c;
}
pair<bint, ll> ldiv(const bint& a, ll b){// c = a / b ; rm = a % b
 bint c;
 ll rm = 0;
 dforn(i, a.l){
      rm = rm * BASE + a.n[i];
      c.n[i] = rm / b;
      rm %= b;
  }
  c.l = a.l;
  c.invar();
  return make_pair(c, rm);
}
bint operator/(const bint&a, ll b){return ldiv(a, b).first;}
ll operator%(const bint&a, ll b){return ldiv(a, b).second;}
pair<bint, bint> ldiv(const bint& a, const bint& b){
 bint c;
  bint rm = 0;
  dforn(i, a.l){
    if (rm.l==1 && !rm.n[0])
      rm.n[0] = a.n[i];
    else{
            dforn(j, rm.l) rm.n[j+1] = rm.n[j];
            rm.n[0] = a.n[i];
            rm.l++;
        }
        ll q = rm.n[b.l] * BASE + rm.n[b.l-1];
        ll u = q / (b.n[b.l-1] + 1);
        ll v = q /  b.n[b.l-1] + 1;
        while (u < v-1){
            ll m = (u+v)/2;
            if (b*m <= rm) u = m;
            else v = m;
        }
        c.n[i]=u;
        rm-=b*u;
    }
  c.l=a.l;
    c.invar();
    return make_pair(c, rm);
}
bint operator/(const bint&a, const bint&b){return ldiv(a, b).first;}
bint operator%(const bint&a, const bint&b){return ldiv(a, b).second;}
```

2.10. HashTables

```
//Compilar: g++ --std=c++11
struct Hash{
  size_t operator()(const ii &a)const{
    size_t s=hash<int>()(a.fst);
    return hash<int>()(a.snd)+0x9e3779b9+(s<<6)+(s>>2);
  }
  size_t operator()(const vector<int> &v)const{
    size_t s=0;
    for(auto &e : v)
      s ^= hash<int>()(e)+0x9e3779b9+(s<<6)+(s>>2);
    return s;
  }
};
unordered_set<ii, Hash> s;
unordered_map<ii, int, Hash> m;//map<key, value, hasher>
```

2.11. Modnum

```
struct mnum{
  static const tipo mod=12582917;
  tipo v;
  mnum(tipo v=0): v(v%mod) {}
  mnum operator+(mnum b){return v+b.v;}
  mnum operator-(mnum b){return v>=b.v? v-b.v : mod-b.v+v;}
  mnum operator*(mnum b){return v*b.v;}
  mnum operator^(int n){
    if(!n) return 1;
    return n%2? (*this)^(n/2)*(this) : (*this)^(n/2);}
};
```

2.12. Treap para set

```
typedef int Key;
typedef struct node *pnode;
struct node{
    Key key;
    int prior, size;
    pnode l,r;
    node(Key key=0): key(key), prior(rand()), size(1), l(0), r(0) {}
};
static int size(pnode p) { return p ? p->size : 0; }
void push(pnode p) {
  // modificar y propagar el dirty a los hijos aca(para lazy)
}
// Update function and size from children's Value
void pull(pnode p) {//recalcular valor del nodo aca (para rmq)
  p->size = 1 + size(p->l) + size(p->r);
}
//junta dos arreglos
pnode merge(pnode l, pnode r) {
  if (!l || !r) return l ? l : r;
  push(l), push(r);
  pnode t;
  if (l->prior < r->prior) l->r=merge(l->r, r), t = l;
  else r->l=merge(l, r->l), t = r;
  pull(t);
  return t;
}
//parte el arreglo en dos, l<key<=r
void split(pnode t, Key key, pnode &l, pnode &r) {
    if (!t) return void(l = r = 0);
    push(t);
    if (key <= t->key) split(t->l, key, l, t->l), r = t;
    else split(t->r, key, t->r, r), l = t;
    pull(t);
}

void erase(pnode &t, Key key) {
    if (!t) return;
    push(t);
    if (key == t->key) t=merge(t->l, t->r);
    else if (key < t->key) erase(t->l, key);
    else erase(t->r, key);
    if(t) pull(t);
}

ostream& operator<<(ostream &out, const pnode &t) {
  if(!t) return out;
    return out << t->l << t->key << '␣' << t->r;
}
pnode find(pnode t, Key key) {
    if (!t) return 0;
    if (key == t->key) return t;
    if (key < t->key) return find(t->l, key);
    return find(t->r, key);
}
struct treap {
    pnode root;
    treap(pnode root=0): root(root) {}
    int size() { return ::size(root); }
    void insert(Key key) {
        pnode t1, t2; split(root, key, t1, t2);
        t1=::merge(t1,new node(key));
        root=::merge(t1,t2);
    }
    void erase(Key key1, Key key2) {
        pnode t1,t2,t3;
        split(root,key1,t1,t2);
        split(t2,key2, t2, t3);
        root=merge(t1,t3);
    }
    void erase(Key key) {::erase(root, key);}
    pnode find(Key key) { return ::find(root, key); }
    Key &operator[](int pos){return find(pos)->key;}//ojito
};
treap merge(treap a, treap b) {return treap(merge(a.root, b.root));}
```

2.13. Treap para arreglo

```
typedef int Value;
typedef struct node *pnode;
struct node{
    Value val;
    int prior, size;
    pnode l,r;
    node(Value val=0): val(val), prior(rand()), size(1), l(0), r(0) {}
};
static int size(pnode p) { return p ? p->size : 0; }
void push(pnode p) {
  // modificar y propagar el dirty a los hijos aca(para lazy)
}
// Update function and size from children's Value
void pull(pnode p) {//recalcular valor del nodo aca (para rmq)
  p->size = 1 + size(p->l) + size(p->r);
}
//junta dos arreglos
pnode merge(pnode l, pnode r) {
  if (!l || !r) return l ? l : r;
  push(l), push(r);
  pnode t;
  if (l->prior < r->prior) l->r=merge(l->r, r), t = l;
  else r->l=merge(l, r->l), t = r;
  pull(t);
  return t;
}
//parte el arreglo en dos, sz(l)==tam
void split(pnode t, int tam, pnode &l, pnode &r) {
  if (!t) return void(l = r = 0);
  push(t);
  if (tam <= size(t->l)) split(t->l, tam, l, t->l), r = t;
  else split(t->r, tam - 1 - size(t->l), t->r, r), l = t;
  pull(t);
}
pnode at(pnode t, int pos) {
  if(!t) exit(1);
  if(pos == size(t->l)) return t;
  if(pos < size(t->l)) return at(t->l, pos);
  return at(t->r, pos - 1 - size(t->l));
}

```

```
ostream& operator<<(ostream &out, const pnode &t) {
  if(!t) return out;
    return out << t->l << t->val << '␣' << t->r;
}


struct arr {//para usar el treap como un arreglo
    pnode root;
    arr(Value val): root(new node(val)) {}
    arr(pnode root=0): root(root) {}
    void insert(int pos, Value val) {//agrega un elemento
        pnode t1,t2; ::split(root, pos, t1, t2);
        t1=merge(t1, new node(val));
        root=merge(t1,t2);
    }
    void erase(int i, int j=-1) {//borra un rango
    if(j==-1) j=i+1;
        pnode t1,t2,t3;
        ::split(root,i,t1,t2);
        ::split(t2,j-i,t2,t3);
        root=merge(t1, t3);
    }
    void push_back(Value val) {root=merge(root, new node(val));}
    Value &operator[](int pos){return at(root, pos)->val;}//ojito
    //parte el arreglo en dos con sz(l)==tam
    void split(int tam, arr &l, arr &r){::split(root, tam, l.root, r.root);}
    //dado un rango, parte el arreglo en tres. m = rango [i, j)
    void split(int i, int j, arr &l, arr &m, arr &r){
    ::split(root, i, l.root, m.root);
    ::split(m.root, j-i, m.root, r.root);
  }
};
//concatena dos arreglos
arr merge(arr a, arr b){return arr(merge(a.root, b.root));}
```

2.14. Convex Hull Trick

```
const ll is_query = -(1LL<<62);
struct Line {
    ll m, b;
    mutable multiset<Line>::iterator it;
    const Line *succ(multiset<Line>::iterator it) const;
    bool operator<(const Line& rhs) const {
        if (rhs.b != is_query) return m < rhs.m;
        const Line *s=succ(it);
        if(!s) return 0;
        ll x = rhs.m;
        return b - s->b < (s->m - m) * x;
    }
};
struct HullDynamic : public multiset<Line>{ // will maintain upper hull for maximum
    bool bad(iterator y) {
        iterator z = next(y);
        if (y == begin()) {
            if (z == end()) return 0;
            return y->m == z->m && y->b <= z->b;
        }
        iterator x = prev(y);
        if (z == end()) return y->m == x->m && y->b <= x->b;
        return (x->b - y->b)*(z->m - y->m) >= (y->b - z->b)*(y->m - x->m);
    }
    iterator next(iterator y){return ++y;}
    iterator prev(iterator y){return --y;}
    void insert_line(ll m, ll b) {
        iterator y = insert((Line) { m, b });
        y->it=y;
        if (bad(y)) { erase(y); return; }
        while (next(y) != end() && bad(next(y))) erase(next(y));
        while (y != begin() && bad(prev(y))) erase(prev(y));
    }
    ll eval(ll x) {
        Line l = *lower_bound((Line) { x, is_query });
        return l.m * x + l.b;
    }
}h;
const Line *Line::succ(multiset<Line>::iterator it) const{
    return (++it==h.end()? NULL : &*it);}
```

2.15. Gain-Cost Set

```
//esta estructura mantiene pairs(beneficio, costo)
//de tal manera que en el set quedan ordenados
//por beneficio Y COSTO creciente. (va borrando los que no son optimos)
struct V{
  int gain, cost;
  bool operator<(const V &b)const{return gain<b.gain;}
};
set<V> s;
void add(V x){
  set<V>::iterator p=s.lower_bound(x);//primer elemento mayor o igual
  if(p!=s.end() && p->cost <= x.cost) return;//ya hay uno mejor
  p=s.upper_bound(x);//primer elemento mayor
  if(p!=s.begin()){//borro todos los peores (<=beneficio  y >=costo)
    --p;//ahora es ultimo elemento menor o igual
    while(p->cost >= x.cost){
      if(p==s.begin()){s.erase(p); break;}
      s.erase(p--);
    }
  }
  s.insert(x);
}
int get(int gain){//minimo costo de obtener tal ganancia
  set<V>::iterator p=s.lower_bound((V){gain, 0});
  return p==s.end()? INF : p->cost;}
```

2.16. Set con busq binaria

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace  __gnu_pbds;
typedef tree<int,null_type,less<int>,//key,mapped type, comparator
    rb_tree_tag,tree_order_statistics_node_update> set_t;
//find_by_order(i) devuelve iterador al i-esimo elemento
//order_of_key(k): devuelve la pos del lower bound de k
//Ej: 12, 100, 505, 1000, 10000.
//order_of_key(10) == 0, order_of_key(100) == 1,
//order_of_key(707) == 3, order_of_key(9999999) == 5
```

3. Algos

3.1. Longest Increasing Subsecuence

```
//Para non-increasing, cambiar comparaciones y revisar busq binaria
//Given an array, paint it in the least number of colors so that each color turns to a non-increasing subsequence.
//Solution:Min number of colors=Length of the longest increasing subsequence
int N, a[MAXN];//secuencia y su longitud
ii d[MAXN+1];//d[i]=ultimo valor de la subsecuencia de tamanio i
int p[MAXN];//padres
vector<int> R;//respuesta
void rec(int i){
  if(i==-1) return;
  R.push_back(a[i]);
  rec(p[i]);
}
int lis(){//O(nlogn)
  d[0] = ii(-INF, -1); forn(i, N) d[i+1]=ii(INF, -1);
  forn(i, N){
    int j = upper_bound(d, d+N+1, ii(a[i], INF))-d;
    if (d[j-1].first < a[i]&&a[i] < d[j].first){
      p[i]=d[j-1].second;
      d[j] = ii(a[i], i);
    }
  }
  R.clear();
  dforn(i, N+1) if(d[i].first!=INF){
    rec(d[i].second);//reconstruir
    reverse(R.begin(), R.end());
    return i;//longitud
  }
  return 0;
}
```

3.2. Manacher

```
int d1[MAXN];//d1[i]=long del maximo palindromo impar con centro en i
int d2[MAXN];//d2[i]=analogo pero para longitud par
//0 1 2 3 4
//a a b c c <--d1[2]=3
//a a b b <--d2[2]=2 (estan uno antes)
void manacher(){
  int l=0, r=-1, n=sz(s);
  forn(i, n){
    int k=(i>r? 1 : min(d1[l+r-i], r-i));
    while(i+k<n && i-k>=0 && s[i+k]==s[i-k]) ++k;
  d1[i] = k--;
  if(i+k > r) l=i-k, r=i+k;
 }
 l=0, r=-1;
 forn(i, n){
  int k=(i>r? 0 : min(d2[l+r-i+1], r-i+1))+1;
  while(i+k-1<n && i-k>=0 && s[i+k-1]==s[i-k]) k++;
  d2[i] = --k;
  if(i+k-1 > r) l=i-k, r=i+k-1;
 }
```

3.3. Alpha-Beta prunning

```
ll alphabeta(State &s, bool player = true, int depth = 1e9, ll alpha = -INF, ll beta = INF) { //player = true -> Maximiza
  if(s.isFinal()) return s.score;
 //~ if (!depth) return s.heuristic();
  vector<State> children;
  s.expand(player, children);
  int n = children.size();
  forn(i, n) {
    ll v = alphabeta(children[i], !player, depth-1, alpha, beta);
    if(!player) alpha = max(alpha, v);
    else beta = min(beta, v);
    if(beta <= alpha) break;
  }
  return !player ? alpha : beta;}
```

4. Strings

4.1. KMP

```
string T;//cadena donde buscar(where)
string P;//cadena a buscar(what)
int b[MAXLEN];//back table
void kmppre(){//by gabina with love
  int i =0, j=-1; b[0]=-1;
  while(i<sz(P)){
    while(j>=0 && P[i] != P[j]) j=b[j];
    i++, j++;
    b[i] = j;
  }
}

void kmp(){
  int i=0, j=0;
  while(i<sz(T)){
    while(j>=0 && T[i]!=P[j]) j=b[j];
    i++, j++;
    if(j==sz(P)){
      printf("P␣is␣found␣at␣index␣%d␣in␣T\n", i-j);
      j=b[j];
    }
  }
}
```

4.2. Trie

```
struct trie{
 map<char, trie> m;
 void add(const string &s, int p=0){
  if(s[p]) m[s[p]].add(s, p+1);
 }
 void dfs(){
  //Do stuff
  forall(it, m)
   it->second.dfs();
 }
};
```

4.3. Suffix Array (largo, nlogn)

```
#define MAX_N 1000
#define rBOUND(x) (x<n? r[x] : 0)
//sa will hold the suffixes in order.
int sa[MAX_N], r[MAX_N], n;
string s; //input string, n=sz(s)

int f[MAX_N], tmpsa[MAX_N];
void countingSort(int k){
 zero(f);
 forn(i, n) f[rBOUND(i+k)]++;
 int sum=0;
 forn(i, max(255, n)){
  int t=f[i]; f[i]=sum; sum+=t;}
 forn(i, n)
  tmpsa[f[rBOUND(sa[i]+k)]++]=sa[i];
  memcpy(sa, tmpsa, sizeof(sa));
}
void constructsa(){//O(n log n)
  n=sz(s);
  forn(i, n) sa[i]=i, r[i]=s[i];
  for(int k=1; k<n; k<<=1){
    countingSort(k), countingSort(0);
    int rank, tmpr[MAX_N];
    tmpr[sa[0]]=rank=0;
    forr(i, 1, n)
      tmpr[sa[i]]=(r[sa[i]]==r[sa[i-1]] && r[sa[i]+k]==r[sa[i-1]+k] )? rank : ++rank;
    memcpy(r, tmpr, sizeof(r));
    if(r[sa[n-1]]==n-1) break;
  }
}
void print(){//for debug
  forn(i, n)
    cout << i << '␣' <<
    s.substr(sa[i], s.find( '$', sa[i])-sa[i]) << endl;}
```

4.4. String Matching With Suffix Array

```
//returns (lowerbound, upperbound) of the search
ii stringMatching(string P){ //O(sz(P)lgn)
  int lo=0, hi=n-1, mid=lo;
  while(lo<hi){
    mid=(lo+hi)/2;
    int res=s.compare(sa[mid], sz(P), P);
    if(res>=0) hi=mid;
    else lo=mid+1;
  }
  if(s.compare(sa[lo], sz(P), P)!=0) return ii(-1, -1);
  ii ans; ans.fst=lo;
  lo=0, hi=n-1, mid;
  while(lo<hi){
    mid=(lo+hi)/2;
    int res=s.compare(sa[mid], sz(P), P);
    if(res>0) hi=mid;
    else lo=mid+1;
  }
  if(s.compare(sa[hi], sz(P), P)!=0) hi--;
  ans.snd=hi;
  return ans;
}
```

4.5. LCP (Longest Common Prefix)

```
//Calculates the LCP between consecutives suffixes in the Suffix Array.
//LCP[i] is the length of the LCP between sa[i] and sa[i-1]
int LCP[MAX_N], phi[MAX_N], PLCP[MAX_N];
void computeLCP(){//O(n)
  phi[sa[0]]=-1;
  forr(i, 1, n) phi[sa[i]]=sa[i-1];
  int L=0;
  forn(i, n){
    if(phi[i]==-1) {PLCP[i]=0; continue;}
    while(s[i+L]==s[phi[i]+L]) L++;
    PLCP[i]=L;
    L=max(L-1, 0);
  }
  forn(i, n) LCP[i]=PLCP[sa[i]];
}
```

4.6. Corasick

```

struct trie{
  map<char, trie> next;
  trie* tran[256];//transiciones del automata
  int idhoja, szhoja;//id de la hoja o 0 si no lo es
  //link lleva al sufijo mas largo, nxthoja lleva al mas largo pero que es hoja
  trie *padre, *link, *nxthoja;
  char pch;//caracter que conecta con padre
  trie(): tran(),  idhoja(), padre(), link() {}
  void insert(const string &s, int id=1, int p=0){//id>0!!!
    if(p<sz(s)){
      trie &ch=next[s[p]];
      tran[(int)s[p]]=&ch;
      ch.padre=this, ch.pch=s[p];
      ch.insert(s, id, p+1);
    }
    else idhoja=id, szhoja=sz(s);
  }
  trie* get_link() {
    if(!link){
      if(!padre) link=this;//es la raiz
      else if(!padre->padre) link=padre;//hijo de la raiz
      else link=padre->get_link()->get_tran(pch);
    }
    return link;
  }
  trie* get_tran(int c) {
    if(!tran[c])
      tran[c] = !padre? this : this->get_link()->get_tran(c);
    return tran[c];
  }
  trie *get_nxthoja(){
    if(!nxthoja) nxthoja = get_link()->idhoja? link : link->nxthoja;
    return nxthoja;
  }
  void print(int p){
    if(idhoja)
      cout << "found␣" << idhoja << "␣␣at␣position␣" << p-szhoja << endl
          ;
    if(get_nxthoja()) get_nxthoja()->print(p);
  }
  void matching(const string &s, int p=0){
    print(p);
    if(p<sz(s)) get_tran(s[p])->matching(s, p+1);
```

5. Geometria

5.1. Punto

```
struct pto{
```

5.2. Line

```
int sgn(ll x){return x<0? -1 : !!x;}
struct line{
  line() {}
  double a,b,c;//Ax+By=C
//pto MUST store float coordinates!
  line(double a, double b, double c):a(a),b(b),c(c){}
  line(pto p, pto q): a(q.y-p.y), b(p.x-q.x), c(a*p.x+b*p.y) {}
  int side(pto p){return sgn(ll(a) * p.x + ll(b) * p.y - c);}
};
bool parallels(line l1, line l2){return abs(l1.a*l2.b-l2.a*l1.b)<EPS;}
pto inter(line l1, line l2){//intersection
  double det=l1.a*l2.b-l2.a*l1.b;
  if(abs(det)<EPS) return pto(INF, INF);//parallels
  return pto(l2.b*l1.c-l1.b*l2.c, l1.a*l2.c-l2.a*l1.c)/det;
}
```

5.3. Segment

```
struct segm{
  pto s,f;
  segm(pto s, pto f):s(s), f(f) {}
  pto closest(pto p) {//use for dist to point
     double l2 = dist_sq(s, f);
     if(l2==0.) return s;
     double t =((p-s)*(f-s))/l2/l2;
     if (t<0.) return s;//not write if is a line
     else if(t>1.)return f;//not write if is a line
     return s+((f-s)*t);
  }
    bool inside(pto p){return abs(dist(s, p)+dist(p, f)-dist(s, f))<EPS
        ;}
};

pto inter(segm s1, segm s2){
  pto r=inter(line(s1.s, s1.f), line(s2.s, s2.f));
    if(s1.inside(r) && s2.inside(r))
        return r;
  return pto(INF, INF);
}
```

5.4. Rectangle

```
struct rect{
  //lower-left and upper-right corners
  pto lw, up;
};
//returns if there's an intersection and stores it in r
bool inter(rect a, rect b, rect &r){
  r.lw=pto(max(a.lw.x, b.lw.x), max(a.lw.y, b.lw.y));
  r.up=pto(min(a.up.x, b.up.x), min(a.up.y, b.up.y));
//check case when only a edge is common
  return r.lw.x<r.up.x && r.lw.y<r.up.y;
}
```

5.5. Polygon Area

```
double area(vector<pto> &p){//O(sz(p))
  double area=0;
  forn(i, sz(p)) area+=p[i]^p[(i+1)%sz(p)];
  //if points are in clockwise order then area is negative
  return abs(area)/2;
}
//Area ellipse = M_PI*a*b where a and b are the semi axis lengths
//Area triangle = sqrt(s*(s-a)(s-b)(s-c)) where s=(a+b+c)/2
```

5.6. Circle

```
vec perp(vec v){return vec(-v.y, v.x);}
line bisector(pto x, pto y){
  line l=line(x, y); pto m=(x+y)/2;
  return line(-l.b, l.a, -l.b*m.x+l.a*m.y);
}
struct Circle{
  pto o;
  double r;
  Circle(pto x, pto y, pto z){
    o=inter(bisector(x, y), bisector(y, z));
    r=dist(o, x);
  }
  pair<pto, pto> ptosTang(pto p){
    pto m=(p+o)/2;
    tipo d=dist(o, m);
    tipo a=r*r/(2*d);
    tipo h=sqrt(r*r-a*a);
    pto m2=o+(m-o)*a/d;
    vec per=perp(m-o)/d;
    return make_pair(m2-per*h, m2+per*h);
  }
};
//finds the center of the circle containing p1 and p2 with radius r
//as there may be two solutions swap p1, p2 to get the other
bool circle2PtsRad(pto p1, pto p2, double r, pto &c){
        double d2=(p1-p2).norm_sq(), det=r*r/d2-0.25;
        if(det<0) return false;
        c=(p1+p2)/2+perp(p2-p1)*sqrt(det);
        return true;
}
#define sqr(a) ((a)*(a))
#define feq(a,b) (fabs((a)-(b))<EPS)
pair<tipo, tipo> ecCuad(tipo a, tipo b, tipo c){//a*x*x+b*x+c=0
  tipo dx = sqrt(b*b-4.0*a*c);
  return make_pair((-b + dx)/(2.0*a),(-b - dx)/(2.0*a));
}
pair<pto, pto> interCL(Circle c, line l){
  bool sw=false;
  if((sw=feq(0,l.b))){
  swap(l.a, l.b);
  swap(c.o.x, c.o.y);
  }
  pair<tipo, tipo> rc = ecCuad(
  sqr(l.a)+sqr(l.b),
  2.0*l.a*l.b*c.o.y-2.0*(sqr(l.b)*c.o.x+l.c*l.a),
  sqr(l.b)*(sqr(c.o.x)+sqr(c.o.y)-sqr(c.r))+sqr(l.c)-2.0*l.c*l.b*c.o.y
  );
  pair<pto, pto> p( pto(rc.first, (l.c - l.a * rc.first) / l.b),
            pto(rc.second, (l.c - l.a * rc.second) / l.b) );
  if(sw){
  swap(p.first.x, p.first.y);
  swap(p.second.x, p.second.y);
  }
  return p;
}
pair<pto, pto> interCC(Circle c1, Circle c2){
  line l;
  l.a = c1.o.x-c2.o.x;
  l.b = c1.o.y-c2.o.y;
  l.c = (sqr(c2.r)-sqr(c1.r)+sqr(c1.o.x)-sqr(c2.o.x)+sqr(c1.o.y)
  -sqr(c2.o.y))/2.0;
  return interCL(c1, l);
}
```

5.7. Point in Poly

```
//checks if v is inside of P, using ray casting
//works with convex and concave.
//excludes boundaries, handle it separately using segment.inside()
bool inPolygon(pto v, vector<pto>& P) {
  bool c = false;
  forn(i, sz(P)){
    int j=(i+1)%sz(P);
  if((P[j].y>v.y) != (P[i].y > v.y) &&
 (v.x < (P[i].x - P[j].x) * (v.y-P[j].y) / (P[i].y - P[j].y) + P[j].x))
   c = !c;
 }
 return c;
}
```

5.8. Convex Check CHECK

```
bool isConvex(vector<int> &p){//O(N)
 int N=sz(p);
 if(N<3) return false;
 bool isLeft=p[0].left(p[1], p[2]);
 forr(i, 1, N)
  if(p[i].left(p[(i+1)%N], p[(i+2)%N])!=isLeft)
   return false;
 return true; }
```

5.9. Convex Hull

```
//stores convex hull of P in S, CCW order
void CH(vector<pto>& P, vector<pto> &S){
 S.clear();
 sort(P.begin(), P.end());
 forn(i, sz(P)){
  while(sz(S)>= 2 && S[sz(S)-1].left(S[sz(S)-2], P[i])) S.pop_back();
  S.pb(P[i]);
 }
 S.pop_back();
 int k=sz(S);
 dforn(i, sz(P)){
  while(sz(S) >= k+2 && S[sz(S)-1].left(S[sz(S)-2], P[i])) S.pop_back();
  S.pb(P[i]);
 }
 S.pop_back();
}
```

5.10. Cut Polygon

```
//cuts polygon Q along the line ab
//stores the left side (swap a, b for the right one) in P
void cutPolygon(pto a, pto b, vector<pto> Q, vector<pto> &P){
 P.clear();
 forn(i, sz(Q)){
  double left1=(b-a)^(Q[i]-a), left2=(b-a)^(Q[(i+1)%sz(Q)]-a);
  if(left1>=0) P.pb(Q[i]);
  if(left1*left2<0)
   P.pb(inter(line(Q[i], Q[(i+1)%sz(Q)]), line(a, b)));
 }
}
```

5.11. Bresenham

```
//plot a line approximation in a 2d map
void bresenham(pto a, pto b){
 pto d=b-a; d.x=abs(d.x), d.y=abs(d.y);
 pto s(a.x<b.x? 1: -1, a.y<b.y? 1: -1);
 int err=d.x-d.y;
 while(1){
  m[a.x][a.y]=1;//plot
  if(a==b) break;
  int e2=2*err;
  if(e2 > -d.y){
   err-=d.y, a.x+=s.x;
  if(e2 < d.x)
   err+= d.x, a.y+= s.y;
 }
}
```

5.12. Rotate Matrix

```
//rotates matrix t 90 degrees clockwise
//using auxiliary matrix t2(faster)
void rotate(){
 forn(x, n) forn(y, n)
  t2[n-y-1][x]=t[x][y];
 memcpy(t, t2, sizeof(t));
}
```

5.13. Interseccion de Circulos en n3log(n)

```
struct event {
  double x; int t;
  event(double xx, int tt) : x(xx), t(tt) {}
  bool operator <(const event &o) const { return x < o.x; }
};
typedef vector<Circle> VC;
typedef vector<event> VE;
int n;
double cuenta(VE &v, double A,double B) {
    sort(v.begin(), v.end());
    double res = 0.0, lx = ((v.empty())?0.0:v[0].x);
    int contador = 0;
    forn(i,sz(v)) {
        // interseccion de todos (contador == n), union de todos (contador > 0),
        // conjunto de puntos cubierto por exacta k Circulos (contador == k)
        if (contador == n) res += v[i].x - lx;
        contador += v[i].t;
        lx = v[i].x;
    }
    return res;
}
// Primitiva de sqrt(r*r - x*x) como funcion double de una variable x.
inline double primitiva(double x,double r) {
    if (x >= r) return r*r*M_PI/4.0;
    if (x <= -r) return -r*r*M_PI/4.0;
    double raiz = sqrt(r*r-x*x);
    return 0.5 * (x * raiz + r*r*atan(x/raiz));
}
double interCircle(VC &v) {
    vector<double> p; p.reserve(v.size() * (v.size() + 2));
    forn(i,sz(v)) {
        p.push_back(v[i].c.x + v[i].r);
        p.push_back(v[i].c.x - v[i].r);
    }
    forn(i,sz(v)) forn(j,i) {
        Circle &a = v[i], b = v[j];
        double d = (a.c - b.c).norm();
        if (fabs(a.r - b.r) < d && d < a.r + b.r) {
            double alfa = acos((sqr(a.r) + sqr(d) - sqr(b.r)) / (2.0 * d * a.r));
            pto vec = (b.c - a.c) * (a.r / d);
            p.pb((a.c + rotate(vec, alfa)).x);
            p.pb((a.c + rotate(vec, -alfa)).x);
        }
    }
    sort(p.begin(), p.end());
    double res = 0.0;
    forn(i,sz(p)-1) {
        const double A = p[i], B = p[i+1];
        VE ve; ve.reserve(2 * v.size());
        forn(j,sz(v)) {
            const Circle &c = v[j];
            double arco = primitiva(B-c.c.x,c.r) - primitiva(A-c.c.x,c.r);
            double base = c.c.y * (B-A);
            ve.push_back(event(base + arco,-1));
            ve.push_back(event(base - arco, 1));
        }
        res += cuenta(ve,A,B);
    }
    return res;
}
```

6. Math

6.1. Identidades

$$\sum_{i=0}^{n} \binom{n}{i} = 2^n$$

$$\sum_{i=0}^{n} i\binom{n}{i} = n * 2^{n-1}$$

$$\sum_{i=m}^{n} i = \frac{n(n+1)}{2} - \frac{m(m-1)}{2} = \frac{(n+1-m)(n+m)}{2}$$

$$\sum_{i=0}^{n} i = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$$\sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6}$$

$$\sum_{i=0}^{n} i(i-1) = \frac{8}{6}(\frac{n}{2})(\frac{n}{2}+1)(n+1) \text{ (doubles)} \rightarrow \text{Sino ver caso impar y par}$$

$$\sum_{i=0}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2 = \frac{n^4}{4} + \frac{n^3}{2} + \frac{n^2}{4} = \left[\sum_{i=1}^{n} i\right]^2$$

$$\sum_{i=0}^{n} i^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30} = \frac{n^5}{5} + \frac{n^4}{2} + \frac{n^3}{3} - \frac{n}{30}$$

$$\sum_{i=0}^{n} i^p = \frac{(n+1)^{p+1}}{p+1} + \sum_{k=1}^{p} \frac{B_k}{p-k+1}\binom{p}{k}(n+1)^{p-k+1}$$

$$r = e - v + k + 1$$

Teorema de Pick: (Area, puntos interiores y puntos en el borde)

$$A = I + \frac{B}{2} - 1$$

6.2. Ec. Caracteristica

$$a_0T(n) + a_1T(n-1) + ... + a_kT(n-k) = 0$$

$$p(x) = a_0x^k + a_1x^{k-1} + ... + a_k$$

Sean $r_1, r_2, ..., r_q$ las raíces distintas, de mult. $m_1, m_2, ..., m_q$

$$T(n) = \sum_{i=1}^{q}\sum_{j=0}^{m_i-1} c_{ij}n^jr_i^n$$

Las constantes c_{ij} se determinan por los casos base.

6.3. Combinatorio

```
forn(i, MAXN+1){//comb[i][k]=i tomados de a k
  comb[i][0]=comb[i][i]=1;
  forr(k, 1, i) comb[i][k]=(comb[i-1][k]+comb[i-1][k-1]) %MOD;
}

ll lucas (ll n, ll k, int p){ //Calcula (n,k) %p teniendo comb[p][p] precalculado.
  ll aux = 1;
  while (n + k){
    aux = (aux * comb[n %p][k %p]) %p;
    n/=p, k/=p;
  }
  return aux;
}
```

6.4. Exp. de Numeros Mod.

```
ll expmod (ll b, ll e, ll m){//O(log b)
  if(!e) return 1;
  ll q= expmod(b,e/2,m); q=(q*q) %m;
  return e %2? (b * q) %m : q;
}
```

6.5. Exp. de Matrices y Fibonacci en log(n)

```
#define SIZE 350
int NN;
void mul(double a[SIZE][SIZE], double b[SIZE][SIZE])
{
    double res[SIZE][SIZE] = {{0}};
    forn(i, NN) forn(j, NN) forn(k, NN) res[i][j]+=a[i][k]*b[k][j];
    forn(i, NN) forn(j, NN) a[i][j]=res[i][j];
}
void powmat(double a[SIZE][SIZE], int n, double res[SIZE][SIZE])
{
    forn(i, NN) forn(j, NN) res[i][j]=(i==j);
    while(n){
        if(n&1) mul(res, a), n--;
        else mul(a, a), n/=2;
    }
}

struct M22{      // |a b|
  tipo a,b,c,d;// |c d|
  M22 operator*(const M22 &p) const {
    return (M22){a*p.a+b*p.c, a*p.b+b*p.d, c*p.a+d*p.c,c*p.b+d*p.d};}
};
M22 operator^(const M22 &p, int n){
  if(!n) return (M22){1, 0, 0, 1};//identidad
  M22 q=p^(n/2); q=q*q;
  return n %2? p * q : q;}

ll fibo(ll n){//calcula el fibonacci enesimo
  M22 mat=(M22){0, 1, 1, 1}^n;
  return mat.a*f0+mat.b*f1;//f0 y f1 son los valores iniciales
}
```

6.6. Matrices y determinante $O(n^3)$

```
struct Mat {
    vector<vector<double> > vec;
    Mat(int n): vec(n, vector<double>(n) ) {}
    Mat(int n, int m): vec(n, vector<double>(m) ) {}
    vector<double> &operator[](int f){return vec[f];}
    const vector<double> &operator[](int f) const {return vec[f];}
    int size() const {return sz(vec);}
    Mat operator+(Mat &b) { ///this de n x m entonces b de n x m
        Mat m(sz(b),sz(b[0]));
        forn(i,sz(vec)) forn(j,sz(vec[0])) m[i][j] = vec[i][j] + b[i][j];
        return m;
    }
    Mat operator*(const Mat &b) { ///this de n x m entonces b de m x t
        int n = sz(vec), m = sz(vec[0]), t = sz(b[0]);
        Mat mat(n,t);
        forn(i,n) forn(j,t) {
            forn(k,m)
                mat[i][j] += vec[i][k] * b[k][j];
        }
        return mat;
    }
    double determinant(){//sacado de e maxx ru
        double det = 1;
        int n = sz(vec);
        Mat m(*this);
        forn(i, n){//para cada columna
            int k = i;
            forr(j, i+1, n)//busco la fila con mayor val abs
                if(abs(m[j][i])>abs(m[k][i]))
                    k = j;
            if(abs(m[k][i])<1e-9) return 0;
            m[i].swap(m[k]);//la swapeo
            if(i!=k) det = -det;
            det *= m[i][i];
            forr(j, i+1, n) m[i][j] /= m[i][i];
            //hago 0 todas las otras filas
            forn(j, n) if (j!= i && abs(m[j][i])>1e-9)
                forr(k, i+1, n) m[j][k]-=m[i][k]*m[j][i];
        }
        return det;
    }
    Mat identidad(int n) {
        Mat m(n);
        forn(i,n) m[i][i] = 1;
        return m;
    }
    Mat transpuesta() {
        Mat m(sz(vec[0]),sz(vec));
        forn(i,sz(vec[0])) forn(j,sz(vec))
            m[i][j] = vec[j][i];
        return m;
    }
    void print() {
        int n = sz(vec), m = sz(vec[0]);
        cout << "***************" << endl;
        forn(i,n){
            forn(j,m) cout << "␣"+!j << vec[i][j];
            cout << endl;
        }
    }
};
```

6.7. Teorema Chino del Resto

$$y = \sum_{j=1}^{n} (x_j * (\prod_{i=1,i\neq j}^{n} m_i)^{-1}_{m_j} * \prod_{i=1,i\neq j}^{n} m_i)$$

6.8. Funciones de primos

```
ll numPrimeFactors (ll n){
  ll rta = 0;
  map<ll,ll> f=fact(n);
  forall(it, f) rta += it->second;
  return rta;
}

ll numDiffPrimeFactors (ll n){
  ll rta = 0;
  map<ll,ll> f=fact(n);
  forall(it, f) rta += 1;
  return rta;
}

ll sumPrimeFactors (ll n){
  ll rta = 0;
  map<ll,ll> f=fact(n);
  forall(it, f) rta += it->first;
  return rta;
}

ll numDiv (ll n){
  ll rta = 1;
  map<ll,ll> f=fact(n);
  forall(it, f) rta *= (it->second + 1);
  return rta;
}

ll sumDiv (ll n){
  ll rta = 1;
  map<ll,ll> f=fact(n);
  forall(it, f) rta *= ((ll)pow((double)it->first, it->second + 1.0)-1)
      / (it->first-1);
  return rta;
}

ll eulerPhi (ll n){ // con criba: O(lg n)
  ll rta = n;
  map<ll,ll> f=fact(n);
  forall(it, f) rta -= rta / it->first;
  return rta;
}

ll eulerPhi2 (ll n){ // O (sqrt n)
  ll r = n;
  forr (i,2,n+1){
    if ((ll)i*i > n)
      break;
    if (n % i == 0){
      while (n%i == 0) n/=i;
      r -= r/i;
    }}
  if (n != 1)
    r-= r/n;
  return r;
}
```

6.9. Phollard's Rho (rolando)

```
ll gcd(ll a, ll b){return a?gcd(b %a, a):b;}

ll mulmod (ll a, ll b, ll c) { //returns (a*b)%c, and minimize overfloor
  ll x = 0, y = a%c;
  while (b > 0){
    if (b % 2 == 1) x = (x+y) % c;
    y = (y*2) % c;
    b /= 2;
  }
  return x % c;
}

ll expmod (ll b, ll e, ll m){//O(log b)
  if(!e) return 1;
  ll q= expmod(b,e/2,m); q=mulmod(q,q,m);
  return e%2? mulmod(b,q,m) : q;
}

bool es_primo_prob (ll n, int a)
{
  if (n == a) return true;
  ll s = 0,d = n-1;
  while (d % 2 == 0) s++,d/=2;

  ll x = expmod(a,d,n);
  if ((x == 1) || (x+1 == n)) return true;

  forn (i, s-1){
    x = mulmod(x, x, n);
    if (x == 1) return false;
    if (x+1 == n) return true;
  }
  return false;
}

bool rabin (ll n){ //devuelve true si n es primo
  if (n == 1) return false;
  const int ar[] = {2,3,5,7,11,13,17,19,23};
  forn (j,9)
    if (!es_primo_prob(n,ar[j]))
      return false;
  return true;
}

ll rho(ll n){
    if( (n & 1) == 0 ) return 2;
    ll x = 2 , y = 2 , d = 1;
    ll c = rand() % n + 1;
    while( d == 1 ){
        x = (mulmod( x , x , n ) + c)%n;
        y = (mulmod( y , y , n ) + c)%n;
        y = (mulmod( y , y , n ) + c)%n;
        if( x - y >= 0 ) d = gcd( x - y , n );
        else d = gcd( y - x , n );
    }
    return d;
}
```

6.10. Criba

```
#define MAXP 100000 //no necesariamente primo
int criba[MAXP+1];
void crearcriba(){
  int w[] = {4,2,4,2,4,6,2,6};
  for(int p=25;p<=MAXP;p+=10) criba[p]=5;
  for(int p=9;p<=MAXP;p+=6) criba[p]=3;
  for(int p=4;p<=MAXP;p+=2) criba[p]=2;
  for(int p=7,cur=0;p*p<=MAXP;p+=w[cur++&7]) if (!criba[p])
```

```
    for(int j=p*p;j<=MAXP;j+=(p<<1)) if(!criba[j]) criba[j]=p;
}

vector<int> primos;
void buscarprimos(){
  crearcriba();
  forr (i,2,MAXP+1) if (!criba[i]) primos.push_back(i);
}

//~ Useful for bit trick:
//~ #define SET(i) ( criba[(i)>>5]|=1<<((i)&31) )
//~ #define INDEX(i) ( (criba[i>>5]>>((i)&31))&1 )
//~ unsigned int criba[MAXP/32+1];
```

6.11. Factorizacion

Sea $n = \prod p_i^{k_i}$, fact(n) genera un map donde a cada p_i le asocia su k_i

```
//factoriza bien numeros hasta MAXP^2
map<ll,ll> fact(ll n){ //O (cant primos)
  map<ll,ll> ret;
  forall(p, primos){
    while(!(n%*p)){
      ret[*p]++;//divisor found
      n/=*p;
    }
  }
  if(n>1) ret[n]++;
  return ret;
}

//factoriza bien numeros hasta MAXP
map<ll,ll> fact2(ll n){ //O (lg n)
  map<ll,ll> ret;
  while (criba[n]){
    ret[criba[n]]++;
    n/=criba[n];
  }
  if(n>1) ret[n]++;
  return ret;
}

map<ll,ll> f3;
void fact3(ll n){ //O (lg n)^3. un solo numero
    if (n == 1) return;
    if (rabin(n))
        f3[n]++;
    else{
        ll aux = rho(n);
        fact3(aux); fact3(n/aux);
    }
  if(n>1) f3[n]++;
  return;
}

//Usar asi:  divisores(fac, divs, fac.begin()); NO ESTA ORDENADO
void divisores(const map<ll,ll> &f, vector<ll> &divs, map<ll,ll>::iterator it, ll n=1){
    if(it==f.begin()) divs.clear();
    if(it==f.end()) {
        if(n>1) divs.pb(n);
        return;
    }
    ll p=it->fst, k=it->snd; ++it;
    forn(_, k+1)
        divisores(f, divs, it, n), n*=p;
}
```

6.12. GCD

```
tipo gcd(tipo a, tipo b){return a?gcd(b %a, a):b;}
```

6.13. Extended Euclid

```
void extendedEuclid (ll a, ll b){ //a * x + b * y = d
  if (!b) { x = 1; y = 0; d = a; return;}
  extendedEuclid (b, a%b);
  ll x1 = y;
  ll y1 = x - (a/b) * y;
  x = x1; y = y1;
}
```

6.14. LCM

```
tipo lcm(tipo a, tipo b){return a / gcd(a,b) * b;}
```

6.15. Inversos

```
#define MAXMOD 15485867
ll inv[MAXMOD];//inv[i]*i=1 mod MOD
void calc(int p){//O(p)
  inv[1]=1;
  forr(i, 2, p) inv[i]= p-((p/i)*inv[p%i])%p;
}
int inverso(int x){//O(log x)
  return expmod(x, eulerphi(MOD)-2);//si mod no es primo(sacar a mano)
  return expmod(x, MOD-2);//si mod es primo
}
```

6.16. Simpson

```
double integral(double a, double b, int n=10000) {//O(n), n=cantdiv
  double area=0, h=(b-a)/n, fa=f(a), fb;
  forn(i, n){
    fb=f(a+h*(i+1));
    area+=fa+ 4*f(a+h*(i+0.5)) +fb, fa=fb;
  }
  return area*h/6.;}
```

6.17. Fraction

```
tipo mcd(tipo a, tipo b){return a?mcd(b%a, a):b;}
struct frac{
  tipo p,q;
  frac(tipo p=0, tipo q=1):p(p),q(q) {norm();}
  void norm(){
    tipo a = mcd(p,q);
    if(a) p/=a, q/=a;
    else q=1;
    if (q<0) q=-q, p=-p;}
  frac operator+(const frac& o){
    tipo a = mcd(q,o.q);
    return frac(p*(o.q/a)+o.p*(q/a), q*(o.q/a));}
  frac operator-(const frac& o){
    tipo a = mcd(q,o.q);
    return frac(p*(o.q/a)-o.p*(q/a), q*(o.q/a));}
  frac operator*(frac o){
    tipo a = mcd(q,o.p), b = mcd(o.q,p);
    return frac((p/b)*(o.p/a), (q/a)*(o.q/b));}
  frac operator/(frac o){
    tipo a = mcd(q,o.q), b = mcd(o.p,p);
    return frac((p/b)*(o.q/a),(q/a)*(o.p/b));}
  bool operator<(const frac &o) const{return p*o.q < o.p*q;}
  bool operator==(frac o){return p==o.p&&q==o.q;}
};
```

6.18. Polinomio

```
struct poly {
    vector<tipo> c;//guarda los coeficientes del polinomio
    poly(const vector<tipo> &c): c(c) {}
    poly() {}
  int gr(){//calculates grade of the polynomial
    return sz(c); }
  bool isnull() {return c.empty();}
    poly operator+(const poly &o) const {
        int m = sz(c), n = sz(o.c);
        vector<tipo> res(max(m,n));
        forn(i, m) res[i] += c[i];
        forn(i, n) res[i] += o.c[i];
        return poly(res);
    }
    poly operator*(const poly &o) const {
        int m = sz(c), n = sz(o.c);
        vector<tipo> res(m+n-1);
        forn(i, m) forn(j, n) res[i+j]+=c[i]*o.c[j];
        return poly(res);
    }
  tipo eval(tipo v) {
    tipo sum = 0;
    dforn(i, sz(c)) sum=sum*v + c[i];
    return sum;
  }
    //poly contains only a vector<int> c (the coeficients)
  //the following function generates the roots of the polynomial
//it can be easily modified to return float roots
  set<tipo> roots(){
    set<tipo> roots;
    tipo a0 = abs(c[0]), an = abs(c[sz(c)-1]);
    vector<tipo> ps,qs;
    forr(p,1,sqrt(a0)+1) if (a0%p==0) ps.pb(p),ps.pb(a0/p);
    forr(q,1,sqrt(an)+1) if (an%q==0) qs.pb(q),qs.pb(an/q);
    forall(pt,ps)
      forall(qt,qs) if ( (*pt) % (*qt)==0 ) {
        tipo root = abs((*pt) / (*qt));
    if (eval(root)==0) roots.insert(root);
   }
  return roots;
 }
};
poly interpolate(const vector<tipo> &x, const vector<tipo> &y) {
  int n = sz(x);
  poly p;
  vector<tipo> aux(2);
  forn(i, n) {
    double a = y[i] - p.eval(x[i]);
    forn(j, i) a /= x[i] - x[j];
    poly add(vector<tipo>(1, a));
    forn(j, i) aux[0]=-x[j], aux[1]=1, add = add*aux;
    p = p + add;
  }
  return p;
}
//the following functions allows parsing an expression like
//34+150+4*45
//into a polynomial(el numero en funcion de la base)
#define LAST(s) (sz(s)? s[sz(s)-1] : 0)
#define POP(s) s.erase(--s.end());
poly D(string &s) {
 poly d;
 for(int i=0; isdigit(LAST(s)); i++) d.c.push_back(LAST(s)-'0'), POP(s);
 return d;}

poly T(string &s) {
 poly t=D(s);
 if (LAST(s)=='*'){POP(s); return T(s)*t;}
 return t;
}
//main function, call this to parse
poly E(string &s) {
 poly e=T(s);
 if (LAST(s)=='+'){POP(s); return E(s)+e;}
 return e;
}
```

6.19. Ec. Lineales

```
bool resolver_ev(Mat a, Vec y, Vec &x, Mat &ev){
 int n = a.size(), m = n?a[0].size():0, rw = min(n, m);
 vector<int> p; forn(i,m) p.push_back(i);
 forn(i, rw) {
  int uc=i, uf=i;
  forr(f, i, n) forr(c, i, m) if(fabs(a[f][c])>fabs(a[uf][uc])) {uf=f;uc=c;}
  if (feq(a[uf][uc], 0)) { rw = i; break; }
  forn(j, n) swap(a[j][i], a[j][uc]);
  swap(a[i], a[uf]); swap(y[i], y[uf]); swap(p[i], p[uc]);
  tipo inv = 1 / a[i][i]; //aca divide
  forr(j, i+1, n) {
   tipo v = a[j][i] * inv;
   forr(k, i, m) a[j][k]-=v * a[i][k];
   y[j] -= v*y[i];
  }
 } // rw = rango(a), aca la matriz esta triangulada
 forr(i, rw, n) if (!feq(y[i],0)) return false; // checkeo de compatibilidad
 x = vector<tipo>(m, 0);
 dforn(i, rw){
  tipo s = y[i];
  forr(j, i+1, rw) s -= a[i][j]*x[p[j]];
  x[p[i]] = s / a[i][i]; //aca divide
 }
 ev = Mat(m-rw, Vec(m, 0)); // Esta parte va SOLO si se necesita el ev
 forn(k, m-rw) {
  ev[k][p[k+rw]] = 1;
  dforn(i, rw){
   tipo s = -a[i][k+rw];
   forr(j, i+1, rw) s -= a[i][j]*ev[k][p[j]];
   ev[k][p[i]] = s / a[i][i]; //aca divide
  }
 }
 return true;
}
```

6.20. FFT

```
//~ typedef complex<double> base; //menos codigo, pero mas lento
//elegir si usar complejos de c (lento) o estos
struct base{
  double r,i;
```

```
    base(double r=0, double i=0):r(r), i(i){}
    double real()const{return r;}
    void operator/=(const int c){r/=c, i/=c;}
};
base operator*(const base &a, const base &b){
    return base(a.r*b.r-a.i*b.i, a.r*b.i+a.i*b.r);}
base operator+(const base &a, const base &b){
    return base(a.r+b.r, a.i+b.i);}
base operator-(const base &a, const base &b){
    return base(a.r-b.r, a.i-b.i);}
vector<int> rev;
vector<base> wlen_pw;
inline static void fft(base a[], int n, bool invert) {
    forn(i, n) if(i<rev[i]) swap(a[i], a[rev[i]]);
  for (int len=2; len<=n; len<<=1) {
    double ang = 2*M_PI/len * (invert?-1:+1);
    int len2 = len>>1;
    base wlen (cos(ang), sin(ang));
    wlen_pw[0] = base (1, 0);
        forr(i, 1, len2) wlen_pw[i] = wlen_pw[i-1] * wlen;
    for (int i=0; i<n; i+=len) {
      base t,
        *pu = a+i,
        *pv = a+i+len2,
        *pu_end = a+i+len2,
        *pw = &wlen_pw[0];
      for (; pu!=pu_end; ++pu, ++pv, ++pw) {
        t = *pv * *pw;
        *pv = *pu - t;
        *pu = *pu + t;
      }
    }
  }
  if (invert) forn(i, n) a[i]/= n;
}
inline static void calc_rev(int n){//precalculo: llamar antes de fft!!
    wlen_pw.resize(n);
    rev.resize(n);
    int lg=31-__builtin_clz(n);
    forn(i, n){
    rev[i] = 0;
        forn(k, lg) if(i&(1<<k))
            rev[i]|=1<<(lg-1-k);
    }
}
inline static void multiply(const vector<int> &a, const vector<int> &b, vector<int> &res) {
  vector<base> fa (a.begin(), a.end()),  fb (b.begin(), b.end());
    int n=1;
  while(n < max(sz(a), sz(b))) n <<= 1;
  n <<= 1;
    calc_rev(n);
  fa.resize (n),  fb.resize (n);
  fft (&fa[0], n, false),  fft (&fb[0], n, false);
  forn(i, n) fa[i] = fa[i] * fb[i];
  fft (&fa[0], n, true);
  res.resize(n);
    forn(i, n) res[i] = int (fa[i].real() + 0.5);
}
void toPoly(const string &s, vector<int> &P){//convierte un numero a polinomio
    P.clear();
    dforn(i, sz(s)) P.pb(s[i]-'0');
}
```

6.21. Tablas y cotas (Primos, Divisores, Factoriales, etc)

Factoriales

$0! = 1$	$11! = 39.916.800$
$1! = 1$	$12! = 479.001.600$ ($\in$ `int`)
$2! = 2$	$13! = 6.227.020.800$
$3! = 6$	$14! = 87.178.291.200$
$4! = 24$	$15! = 1.307.674.368.000$
$5! = 120$	$16! = 20.922.789.888.000$
$6! = 720$	$17! = 355.687.428.096.000$
$7! = 5.040$	$18! = 6.402.373.705.728.000$
$8! = 40.320$	$19! = 121.645.100.408.832.000$
$9! = 362.880$	$20! = 2.432.902.008.176.640.000$ ($\in$ `tint`)
$10! = 3.628.800$	$21! = 51.090.942.171.709.400.000$

max signed tint = 9.223.372.036.854.775.807

max unsigned tint = 18.446.744.073.709.551.615

Primos

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197 199 211 223 227 229 233 239 241 251 257 263 269 271 277 281 283 293 307 311 313 317 331 337 347 349 353 359 367 373 379 383 389 397 401 409 419 421 431 433 439 443 449 457 461

463 467 479 487 491 499 503 509 521 523 541 547 557 563 569 571 577 587 593 599
601 607 613 617 619 631 641 643 647 653 659 661 673 677 683 691 701 709 719 727
733 739 743 751 757 761 769 773 787 797 809 811 821 823 827 829 839 853 857 859
863 877 881 883 887 907 911 919 929 937 941 947 953 967 971 977 983 991 997 1009
1013 1019 1021 1031 1033 1039 1049 1051 1061 1063 1069 1087 1091 1093 1097 1103
1109 1117 1123 1129 1151 1153 1163 1171 1181 1187 1193 1201 1213 1217 1223 1229
1231 1237 1249 1259 1277 1279 1283 1289 1291 1297 1301 1303 1307 1319 1321 1327
1361 1367 1373 1381 1399 1409 1423 1427 1429 1433 1439 1447 1451 1453 1459 1471
1481 1483 1487 1489 1493 1499 1511 1523 1531 1543 1549 1553 1559 1567 1571 1579
1583 1597 1601 1607 1609 1613 1619 1621 1627 1637 1657 1663 1667 1669 1693 1697
1699 1709 1721 1723 1733 1741 1747 1753 1759 1777 1783 1787 1789 1801 1811 1823
1831 1847 1861 1867 1871 1873 1877 1879 1889 1901 1907 1913 1931 1933 1949 1951
1973 1979 1987 1993 1997 1999 2003 2011 2017 2027 2029 2039 2053 2063 2069 2081

Primos cercanos a 10^n

9941 9949 9967 9973 10007 10009 10037 10039 10061 10067 10069 10079
99961 99971 99989 99991 100003 100019 100043 100049 100057 100069
999959 999961 999979 999983 1000003 1000033 1000037 1000039
9999943 9999971 9999973 9999991 10000019 10000079 10000103 10000121
99999941 99999959 99999971 99999989 100000007 100000037 100000039 100000049
999999893 999999929 999999937 1000000007 1000000009 1000000021 1000000033

Cantidad de primos menores que 10^n

$\pi(10^1) = 4$; $\pi(10^2) = 25$; $\pi(10^3) = 168$; $\pi(10^4) = 1229$; $\pi(10^5) = 9592$
$\pi(10^6) = 78.498$; $\pi(10^7) = 664.579$; $\pi(10^8) = 5.761.455$; $\pi(10^9) = 50.847.534$
$\pi(10^{10}) = 455.052{,}511$; $\pi(10^{11}) = 4.118.054.813$; $\pi(10^{12}) = 37.607.912.018$

Divisores

Cantidad de divisores (σ_0) para *algunos* $n / \neg\exists n' < n, \sigma_0(n') \geqslant \sigma_0(n)$
$\sigma_0(60) = 12$; $\sigma_0(120) = 16$; $\sigma_0(180) = 18$; $\sigma_0(240) = 20$; $\sigma_0(360) = 24$
$\sigma_0(720) = 30$; $\sigma_0(840) = 32$; $\sigma_0(1260) = 36$; $\sigma_0(1680) = 40$; $\sigma_0(10080) = 72$
$\sigma_0(15120) = 80$; $\sigma_0(50400) = 108$; $\sigma_0(83160) = 128$; $\sigma_0(110880) = 144$
$\sigma_0(498960) = 200$; $\sigma_0(554400) = 216$; $\sigma_0(1081080) = 256$; $\sigma_0(1441440) = 288$
$\sigma_0(4324320) = 384$; $\sigma_0(8648640) = 448$
Suma de divisores (σ_1) para *algunos* $n / \neg\exists n' < n, \sigma_1(n') \geqslant \sigma_1(n)$
$\sigma_1(96) = 252$; $\sigma_1(108) = 280$; $\sigma_1(120) = 360$; $\sigma_1(144) = 403$; $\sigma_1(168) = 480$
$\sigma_1(960) = 3048$; $\sigma_1(1008) = 3224$; $\sigma_1(1080) = 3600$; $\sigma_1(1200) = 3844$
$\sigma_1(4620) = 16128$; $\sigma_1(4680) = 16380$; $\sigma_1(5040) = 19344$; $\sigma_1(5760) = 19890$
$\sigma_1(8820) = 31122$; $\sigma_1(9240) = 34560$; $\sigma_1(10080) = 39312$; $\sigma_1(10920) = 40320$
$\sigma_1(32760) = 131040$; $\sigma_1(35280) = 137826$; $\sigma_1(36960) = 145152$; $\sigma_1(37800) = 148800$
$\sigma_1(60480) = 243840$; $\sigma_1(64680) = 246240$; $\sigma_1(65520) = 270816$; $\sigma_1(70560) = 280098$
$\sigma_1(95760) = 386880$; $\sigma_1(98280) = 403200$; $\sigma_1(100800) = 409448$
$\sigma_1(491400) = 2083200$; $\sigma_1(498960) = 2160576$; $\sigma_1(514080) = 2177280$
$\sigma_1(982800) = 4305280$; $\sigma_1(997920) = 4390848$; $\sigma_1(1048320) = 4464096$
$\sigma_1(4979520) = 22189440$; $\sigma_1(4989600) = 22686048$; $\sigma_1(5045040) = 23154768$
$\sigma_1(9896040) = 44323200$; $\sigma_1(9959040) = 44553600$; $\sigma_1(9979200) = 45732192$

7. Grafos

7.1. Dijkstra

```
#define INF 1e9
int N;
#define MAX_V 250001
vector<ii> G[MAX_V];
//To add an edge use
#define add(a, b, w) G[a].pb(make_pair(w, b))

ll dijkstra(int s, int t){//O(|E| log |V|)
  priority_queue<ii, vector<ii>, greater<ii> > Q;
  vector<ll> dist(N, INF); vector<int> dad(N, -1);
  Q.push(make_pair(0, s)); dist[s] = 0;
  while(sz(Q)){
    ii p = Q.top(); Q.pop();
    if(p.snd == t) break;
    forall(it, G[p.snd])
      if(dist[p.snd]+it->first < dist[it->snd]){
        dist[it->snd] = dist[p.snd] + it->fst;
        dad[it->snd] = p.snd;
        Q.push(make_pair(dist[it->snd], it->snd));
      }
  }
  return dist[t];
  if(dist[t]<INF)//path generator
    for(int i=t; i!=-1; i=dad[i])
      printf("%d%c", i, (i==s?'\n':'␣'));
}
```

7.2. Bellman-Ford

```
vector<ii> G[MAX_N];//ady. list with pairs (weight, dst)
int dist[MAX_N];
void bford(int src){//O(VE)
  dist[src]=0;
  forn(i, N-1) forn(j, N) if(dist[j]!=INF) forall(it, G[j])
```

```
    dist[it->snd]=min(dist[it->snd], dist[j]+it->fst);
}

bool hasNegCycle(){
  forn(j, N) if(dist[j]!=INF) forall(it, G[j])
    if(dist[it->snd]>dist[j]+it->fst) return true;
  //inside if: all points reachable from it->snd will have -INF distance (do bfs)
  return false;
}
```

7.3. Floyd-Warshall

```
//G[i][j] contains weight of edge (i, j) or INF
//G[i][i]=0
int G[MAX_N][MAX_N];
void floyd(){//O(N^3)
forn(k, N) forn(i, N) if(G[i][k]!=INF) forn(j, N) if(G[k][j]!=INF)
  G[i][j]=min(G[i][j], G[i][k]+G[k][j]);
}
bool inNegCycle(int v){
  return G[v][v]<0;}
//checks if there's a neg. cycle in path from a to b
bool hasNegCycle(int a, int b){
  forn(i, N) if(G[a][i]!=INF && G[i][i]<0 && G[i][b]!=INF)
    return true;
  return false;
}
```

7.4. Kruskal

```
struct Ar{int a,b,w;};
bool operator<(const Ar& a, const Ar &b){return a.w<b.w;}
vector<Ar> E;
ll kruskal(){
    ll cost=0;
    sort(E.begin(), E.end());//ordenar aristas de menor a mayor
    uf.init(n);
    forall(it, E){
        if(uf.comp(it->a)!=uf.comp(it->b)){//si no estan conectados
            uf.unir(it->a, it->b);//conectar
            cost+=it->w;
        }
    }
    return cost;
}
```

7.5. Prim

```
bool taken[MAXN];
priority_queue<ii, vector<ii>, greater<ii> > pq;//min heap
void process(int v){
    taken[v]=true;
    forall(e, G[v])
        if(!taken[e->second]) pq.push(*e);
}

ll prim(){
    zero(taken);
    process(0);
    ll cost=0;
    while(sz(pq)){
        ii e=pq.top(); pq.pop();
        if(!taken[e.second]) cost+=e.first, process(e.second);
    }
    return cost;
}
```

7.6. 2-SAT + Tarjan SCC

```
//We have a vertex representing a var and other for his negation.
//Every edge stored in G represents an implication. To add an equation of the form a||b, use addor(a, b)
//MAX=max cant var, n=cant var
#define addor(a, b) (G[neg(a)].pb(b), G[neg(b)].pb(a))
vector<int> G[MAX*2];
//idx[i]=index assigned in the dfs
//lw[i]=lowest index(closer from the root) reachable from i
int lw[MAX*2], idx[MAX*2], qidx;
stack<int> q;
int qcmp, cmp[MAX*2];
//verdad[cmp[i]]=valor de la variable i
bool verdad[MAX*2+1];

int neg(int x) { return x>=n? x-n : x+n;}
void tjn(int v){
  lw[v]=idx[v]=++qidx;
  q.push(v), cmp[v]=-2;
  forall(it, G[v]){
    if(!idx[*it] || cmp[*it]==-2){
      if(!idx[*it]) tjn(*it);
      lw[v]=min(lw[v], lw[*it]);
    }
  }
  if(lw[v]==idx[v]){
    qcmp++;
    int x;
    do{x=q.top(); q.pop(); cmp[x]=qcmp;}while(x!=v);
    verdad[qcmp]=(cmp[neg(v)]<0);
  }
}
//remember to CLEAR G!!!
bool satisf(){//O(n)
  memset(idx, 0, sizeof(idx)), qidx=0;
  memset(cmp, -1, sizeof(cmp)), qcmp=0;
  forn(i, n){
    if(!idx[i]) tjn(i);
    if(!idx[neg(i)]) tjn(neg(i));
  }
  forn(i, n) if(cmp[i]==cmp[neg(i)]) return false;
  return true;
}
```

7.7. Articulation Points

```
int N;
vector<int> G[1000000];
//V[i]=node number(if visited), L[i]= lowest V[i] reachable from i
int qV, V[1000000], L[1000000], P[1000000];
void dfs(int v, int f){
  L[v]=V[v]=++qV;
  forall(it, G[v])
    if(!V[*it]){
      dfs(*it, v);
      L[v] = min(L[v], L[*it]);
      P[v]+= L[*it]>=V[v];
    }
    else if(*it!=f)
      L[v]=min(L[v], V[*it]);
}
int cantart(){ //O(n)
  qV=0;
  zero(V), zero(P);
  dfs(1, 0); P[1]--;
  int q=0;
  forn(i, N) if(P[i]) q++;
return q;
}
```

7.8. Comp. Biconexas y Puentas

```
struct edge {
  int u,v, comp;
  bool bridge;
};
vector<edge> e;
void addEdge(int u, int v) {
  G[u].pb(sz(e)), G[v].pb(sz(e));
  e.pb((edge){u,v,-1,false});
}
//d[i]=id de la dfs
//b[i]=lowest id reachable from i
int d[MAXN], b[MAXN], t;
int nbc;//cant componentes
int comp[MAXN];//comp[i]=cant comp biconexas a la cual pertenece i
void initDfs(int n) {
  zero(G), zero(comp);
  e.clear();
  forn(i,n) d[i]=-1;
  nbc = t = 0;
}
stack<int> st;
void dfs(int u, int pe) {//O(n + m)
  b[u] = d[u] = t++;
  comp[u] = (pe != -1);
  forall(ne, G[u]) if (*ne != pe){
    int v = e[*ne].u ^ e[*ne].v ^ u;
    if (d[v] == -1) {
      st.push(*ne);
      dfs(v,*ne);
      if (b[v] > d[u]){
        e[*ne].bridge = true; // bridge
      }
      if (b[v] >= d[u]){ // art
    int last;
    do {
     last = st.top(); st.pop();
     e[last].comp = nbc;
    } while (last != *ne);
    nbc++;
    comp[u]++;
   }
   b[u] = min(b[u], b[v]);
  }
  else if (d[v] < d[u]) { // back edge
   st.push(*ne);
   b[u] = min(b[u], d[v]);
  }
 }
}
```

7.9. LCA + Climb

```
const int MAXN=100001;
const int LOGN=20;
//f[v][k] holds the 2^k father of v
//L[v] holds the level of v
int N, f[MAXN][LOGN], L[MAXN];
//call before build:
void dfs(int v, int fa=-1, int lvl=0){//generate required data
 f[v][0]=fa, L[v]=lvl;
 forall(it, G[v])if(*it!=fa)
  dfs(*it, v, lvl+1);
}
void build(){//f[i][0] must be filled previously, O(nlgn)
 forn(k, LOGN-1) forn(i, N) f[i][k+1]=f[f[i][k]][k];}

#define lg(x) (31-__builtin_clz(x))//=floor(log2(x))

int climb(int a, int d){//O(lgn)
 if(!d) return a;
 dforn(i, lg(L[a])+1)
  if(1<<i<=d)
   a=f[a][i], d-=1<<i;
  return a;
}
int lca(int a, int b){//O(lgn)
 if(L[a]<L[b]) swap(a, b);
 a=climb(a, L[a]-L[b]);
 if(a==b) return a;
 dforn(i, lg(L[a])+1)
  if(f[a][i]!=f[b][i])
   a=f[a][i], b=f[b][i];
 return f[a][0];
}
int dist(int a, int b) {//returns distance between nodes
 return L[a]+L[b]-2*L[lca(a, b)];}
```

7.10. Heavy Light Decomposition

```
int treesz[MAXN];//cantidad de nodos en el subarbol del nodo v
int dad[MAXN];//dad[v]=padre del nodo v
void dfs1(int v, int p=-1){//pre-dfs
 dad[v]=p;
 treesz[v]=1;
 forall(it, G[v]) if(*it!=p){
  dfs1(*it, v);
  treesz[v]+=treesz[*it];
 }
}
//PONER Q EN 0  !!!!!
int pos[MAXN], q;//pos[v]=posicion del nodo v en el recorrido de la dfs
//Las cadenas aparecen continuas en el recorrido!
int cantcad;
int homecad[MAXN];//dada una cadena devuelve su nodo inicial
int cad[MAXN];//cad[v]=cadena a la que pertenece el nodo
void heavylight(int v, int cur=-1){
 if(cur==-1) homecad[cur=cantcad++]=v;
 pos[v]=q++;
 cad[v]=cur;
 int mx=-1;
 forn(i, sz(G[v])) if(G[v][i]!=dad[v])
  if(mx==-1 || treesz[G[v][mx]]<treesz[G[v][i]]) mx=i;
 if(mx!=-1) heavylight(G[v][mx], cur);
 forn(i, sz(G[v])) if(i!=mx && G[v][i]!=dad[v])
  heavylight(G[v][i], -1);
}
//ejemplo de obtener el maximo numero en el camino entre dos nodos
//RTA: max(query(low, u), query(low, v)), con low=lca(u, v)
//esta funcion va trepando por las cadenas
```

```
int query(int an, int v){//O(logn)
  //si estan en la misma cadena:
  if(cad[an]==cad[v]) return rmq.get(pos[an], pos[v]+1);
  return max(query(an, dad[homecad[cad[v]]]),
         rmq.get(pos[homecad[cad[v]]], pos[v]+1));
}
```

7.11. Centroid Decomposition

```
int n;
vector<int> G[MAXN];
bool taken[MAXN];//poner todos en FALSE al principio!!
int padre[MAXN];//padre de cada nodo en el centroid tree

int szt[MAXN];
void calcsz(int v, int p) {
  szt[v] = 1;
  forall(it,G[v]) if (*it!=p && !taken[*it])
    calcsz(*it,v), szt[v]+=szt[*it];
}
void centroid(int v=0, int f=-1, int lvl=0, int tam=-1) {//O(nlogn)
  if(tam==-1) calcsz(v, -1), tam=szt[v];
  forall(it, G[v]) if(!taken[*it] && szt[*it]>=tam/2)
    {szt[v]=0; centroid(*it, f, lvl, tam); return;}
  taken[v]=true;
  padre[v]=f;
  forall(it, G[v]) if(!taken[*it])
    centroid(*it, v, lvl+1, -1);
}
```

7.12. Euler Cycle

```
int n,m,ars[MAXE], eq;
vector<int> G[MAXN];//fill G,n,m,ars,eq
list<int> path;
int used[MAXN];
bool usede[MAXE];
queue<list<int>::iterator> q;
int get(int v){
  while(used[v]<sz(G[v]) && usede[ G[v][used[v]] ]) used[v]++;
  return used[v];
}
void explore(int v, int r, list<int>::iterator it){
  int ar=G[v][get(v)]; int u=v^ars[ar];
  usede[ar]=true;
  list<int>::iterator it2=path.insert(it, u);
  if(u!=r) explore(u, r, it2);
  if(get(v)<sz(G[v])) q.push(it);
}
void euler(){
  zero(used), zero(usede);
  path.clear();
  q=queue<list<int>::iterator>();
  path.push_back(0); q.push(path.begin());
  while(sz(q)){
    list<int>::iterator it=q.front(); q.pop();
    if(used[*it]<sz(G[*it])) explore(*it, *it, it);
  }
  reverse(path.begin(), path.end());
}
void addEdge(int u, int v){
  G[u].pb(eq), G[v].pb(eq);
  ars[eq++]=u^v;
}
```

7.13. Diametro árbol

```
vector<int> G[MAXN];
int n,m;

int p[MAXN],d[MAXN],d2[MAXN];
int bfs(int r, int *d) {
  queue<int> q;
  d[r]=0; q.push(r);
  int v;
  while(sz(q)) { v=q.front(); q.pop();
    forall(it,G[v]) if (d[*it]==-1) {
      d[*it]=d[v]+1, p[*it]=v;
      q.push(*it);
    }
  }
  return v;//ultimo nodo visitado
}

vector<int> diams;
vector<ii> centros;
void diametros(){
  memset(d,-1,sizeof(d));
  memset(d2,-1,sizeof(d2));
  diams.clear(), centros.clear();
  forn(i, n) if(d[i]==-1){
    int v,c;
    c=v=bfs(bfs(i, d2), d);
    forn(_,d[v]/2) c=p[c];
    diams.pb(d[v]);
    if(d[v]&1) centros.pb(ii(c, p[c]));
    else centros.pb(ii(c, c));
  }
}
```

7.14. Chu-liu

```
void visit(graph &h, int v, int s, int r,
  vector<int> &no, vector< vector<int> > &comp,
  vector<int> &prev, vector< vector<int> > &next, vector<weight> &mcost,
  vector<int> &mark, weight &cost, bool &found) {
  if (mark[v]) {
    vector<int> temp = no;
    found = true;
    do {
      cost += mcost[v];
      v = prev[v];
      if (v != s) {
        while (comp[v].size() > 0) {
          no[comp[v].back()] = s;
          comp[s].push_back(comp[v].back());
          comp[v].pop_back();
        }
      }
    } while (v != s);
    forall(j,comp[s]) if (*j != r) forall(e,h[*j])
      if (no[e->src] != s) e->w -= mcost[ temp[*j] ];
  }
  mark[v] = true;
  forall(i,next[v]) if (no[*i] != no[v] && prev[no[*i]] == v)
    if (!mark[no[*i]] || *i == s)
      visit(h, *i, s, r, no, comp, prev, next, mcost, mark, cost, found);
}
weight minimumSpanningArborescence(const graph &g, int r) {
    const int n=sz(g);
  graph h(n);
  forn(u,n) forall(e,g[u]) h[e->dst].pb(*e);
  vector<int> no(n);
  vector<vector<int> > comp(n);
  forn(u, n) comp[u].pb(no[u] = u);
  for (weight cost = 0; ;) {
    vector<int> prev(n, -1);
    vector<weight> mcost(n, INF);
    forn(j,n) if (j != r) forall(e,h[j])
      if (no[e->src] != no[j])
        if (e->w < mcost[ no[j] ])
          mcost[ no[j] ] = e->w, prev[ no[j] ] = no[e->src];
    vector< vector<int> > next(n);
    forn(u,n) if (prev[u] >= 0)
      next[ prev[u] ].push_back(u);
    bool stop = true;
    vector<int> mark(n);
    forn(u,n) if (u != r && !mark[u] && !comp[u].empty()) {
      bool found = false;
      visit(h, u, u, r, no, comp, prev, next, mcost, mark, cost, found);
      if (found) stop = false;
    }
    if (stop) {
      forn(u,n) if (prev[u] >= 0) cost += mcost[u];
      return cost;
    }
  }
}
```

7.15. Hungarian

```
#define MAXN 256
#define INFTO 0x7f7f7f7f
int n;
int mt[MAXN][MAXN]; // Matriz de costos (X * Y)
int xy[MAXN], yx[MAXN]; // Matching resultante (X->Y, Y->X)
int lx[MAXN], ly[MAXN], slk[MAXN], slkx[MAXN], prv[MAXN];
char S[MAXN], T[MAXN];
void updtree(int x) {
  forn(y, n) if (lx[x] + ly[y] - mt[x][y] < slk[y]) {
    slk[y] = lx[x] + ly[y] - mt[x][y];
    slkx[y] = x;
} }
int hungar(){//Matching maximo de mayor costo en grafos dirigidos (N^3)
  forn(i, n) {
    ly[i] = 0;
    lx[i] = *max_element(mt[i], mt[i]+n); }
  memset(xy, -1, sizeof(xy));
  memset(yx, -1, sizeof(yx));
  forn(m, n) {
    memset(S, 0, sizeof(S));
    memset(T, 0, sizeof(T));
    memset(prv, -1, sizeof(prv));
    memset(slk, 0x7f, sizeof(slk));
    queue<int> q;
#define bpone(e, p) { q.push(e); prv[e] = p; S[e] = 1; updtree(e); }
    forn(i, n) if (xy[i] == -1) { bpone(i, -2); break; }
    int x=0, y=-1;
    while (y==-1) {
      while (!q.empty() && y==-1) {
        x = q.front(); q.pop();
        forn(j, n) if (mt[x][j] == lx[x] + ly[j] && !T[j]) {
          if (yx[j] == -1) { y = j; break; }
          T[j] = 1;
          bpone(yx[j], x);
        }
      }
      if (y!=-1) break;
      int dlt = INFTO;
      forn(j, n) if (!T[j]) dlt = min(dlt, slk[j]);
      forn(k, n) {
        if (S[k]) lx[k] -= dlt;
        if (T[k]) ly[k] += dlt;
        if (!T[k]) slk[k] -= dlt;
      }
      forn(j, n) if (!T[j] && !slk[j]) {
        if (yx[j] == -1) {
          x = slkx[j]; y = j; break;
        } else {
          T[j] = 1;
          if (!S[yx[j]]) bpone(yx[j], slkx[j]);
        }
      }
    }
    if (y!=-1) {
      for(int p = x; p != -2; p = prv[p]) {
        yx[y] = p;
        int ty = xy[p]; xy[p] = y; y = ty;
      }
    } else break;
  }
  int res = 0;
  forn(i, n) res += mt[i][xy[i]];
  return res;
}
```

8. Network Flow

8.1. Dinic

```

const int MAX = 300;
int nodes, src, dst;
int dist[MAX], q[MAX], work[MAX];

struct Edge {
    int to, rev;
    ll f, cap;
    Edge(int to, int rev, ll f, ll cap) : to(to), rev(rev), f(f), cap(
        cap) {}
};
vector<Edge> G[MAX];

// Adds bidirectional edge
void addEdge(int s, int t, ll cap){
    G[s].pb(Edge(t, sz(G[t]), 0, cap));
    G[t].pb(Edge(s, sz(G[s])-1, 0, 0));
}

bool dinic_bfs(){
    fill(dist, dist+nodes, -1);
    dist[src]=0;
    int qt=0;
    q[qt++]=src;
    for(int qh=0; qh<qt; qh++){
        int u =q[qh];
        forall(e, G[u]){
            int v=e->to;
            if(dist[v]<0 && e->f < e->cap){
                dist[v]=dist[u]+1;
                q[qt++]=v;
            }
        }
    }
    return dist[dst]>=0;
}

ll dinic_dfs(int u, ll f){
    if(u==dst) return f;
    for(int &i=work[u]; i<sz(G[u]); i++){
        Edge &e = G[u][i];
        if(e.cap<=e.f) continue;
        int v=e.to;
        if(dist[v]==dist[u]+1){
                ll df=dinic_dfs(v, min(f, e.cap-e.f));
                if(df>0){
                        e.f+=df;
                        G[v][e.rev].f-= df;
                        return df;
                }
        }
    }
    return 0;
}

ll maxFlow(int _src, int _dst){
    src=_src;
    dst=_dst;
    ll result=0;
    while(dinic_bfs()){
        fill(work, work+nodes, 0);
        while(ll delta=dinic_dfs(src,INF))
            result+=delta;
    }
    // todos los nodos con dist[v]!=-1 vs los que tienen dist[v]==-1 forman el min-cut
    return result;
}
```

8.2. Konig

```
// asume que el dinic YA ESTA tirado
// asume que nodes-1 y nodes-2 son la fuente y destino
int match[maxnodes]; // match[v]=u si u-v esta en el matching, -1 si v no esta matcheado
int s[maxnodes]; // numero de la bfs del koning
queue<int> kq;
// s[e]%2==1  o si e esta en V1 y s[e]==-1-> lo agarras
void koning() {//O(n)
  forn(v,nodes-2) s[v] = match[v] = -1;
  forn(v,nodes-2) forall(it,g[v]) if (it->to < nodes-2 && it->f>0)
    { match[v]=it->to; match[it->to]=v;}
  forn(v,nodes-2) if (match[v]==-1) {s[v]=0;kq.push(v);}
  while(!kq.empty()) {
    int e = kq.front(); kq.pop();
    if (s[e]%2==1) {
      s[match[e]] = s[e]+1;
      kq.push(match[e]);
    } else {

      forall(it,g[e]) if (it->to < nodes-2 && s[it->to]==-1) {
        s[it->to] = s[e]+1;
        kq.push(it->to);
      }
    }
  }
}
```

8.3. Edmonds Karp's

```
#define MAX_V 1000
#define INF 1e9
//special nodes
#define SRC 0
#define SNK 1
map<int, int> G[MAX_V];//limpiar esto
//To add an edge use
#define add(a, b, w) G[a][b]=w
int f, p[MAX_V];
void augment(int v, int minE){
  if(v==SRC) f=minE;
  else if(p[v]!=-1){
    augment(p[v], min(minE, G[p[v]][v]));
    G[p[v]][v]-=f, G[v][p[v]]+=f;
  }
}
ll maxflow(){//O(VE^2)
  ll Mf=0;
  do{
    f=0;
    char used[MAX_V]; queue<int> q; q.push(SRC);
    zero(used), memset(p, -1, sizeof(p));
    while(sz(q)){
      int u=q.front(); q.pop();
      if(u==SNK) break;
      forall(it, G[u])
        if(it->snd>0 && !used[it->fst])
          used[it->fst]=true, q.push(it->fst), p[it->fst]=u;
    }
    augment(SNK, INF);
    Mf+=f;
  }while(f);
  return Mf;
}
```

8.4. Push-Relabel O(N3)

```
#define MAX_V 1000
int N;//valid nodes are [0...N-1]
#define INF 1e9
//special nodes
#define SRC 0
#define SNK 1
map<int, int> G[MAX_V];
//To add an edge use
#define add(a, b, w) G[a][b]=w
ll excess[MAX_V];
int height[MAX_V], active[MAX_V], count[2*MAX_V+1];
queue<int> Q;
void enqueue(int v) {
  if (!active[v] && excess[v] > 0) active[v]=true, Q.push(v); }
void push(int a, int b) {
  int amt = min(excess[a], ll(G[a][b]));
  if(height[a] <= height[b] || amt == 0) return;
  G[a][b]-=amt, G[b][a]+=amt;
  excess[b] += amt, excess[a] -= amt;
  enqueue(b);
}
void gap(int k) {
  forn(v, N){
    if (height[v] < k) continue;
    count[height[v]]--;
    height[v] = max(height[v], N+1);
    count[height[v]]++;
    enqueue(v);
  }
}
void relabel(int v) {
  count[height[v]]--;
  height[v] = 2*N;
  forall(it, G[v])
    if(it->snd)
      height[v] = min(height[v], height[it->fst] + 1);
  count[height[v]]++;
  enqueue(v);
}
ll maxflow() {//O(V^3)
  zero(height), zero(active), zero(count), zero(excess);
  count[0] = N-1;
  count[N] = 1;
  height[SRC] = N;
  active[SRC] = active[SNK] = true;
  forall(it, G[SRC]){
    excess[SRC] += it->snd;
    push(SRC, it->fst);
  }
  while(sz(Q)) {
    int v = Q.front(); Q.pop();
    active[v]=false;
  forall(it, G[v]) push(v, it->fst);
  if(excess[v] > 0)
    count[height[v]] == 1? gap(height[v]):relabel(v);
  }
  ll mf=0;
  forall(it, G[SRC]) mf+=G[it->fst][SRC];
  return mf;
}
```

8.5. Min-cost Max-flow

```
const int MAXN=10000;
const ll INF = 1e14;

struct edge {
  int u, v;
  ll cap, cost, flow;
  ll rem() { return cap - flow; }
};
int nodes;//numero de nodos
vector<int> G[MAXN];
vector<edge> e;
void addEdge(int u, int v, ll cap, ll cost) {
  G[u].pb(sz(e)); e.pb((edge){u,v,cap,cost,0});
  G[v].pb(sz(e)); e.pb((edge){v,u,0,-cost,0});
}
ll pot[MAXN], dist[MAXN], pre[MAXN], cap[MAXN];
ll mxFlow, mnCost;
void flow(int s, int t) {
  fill(pot, pot+nodes, 0);
  mxFlow=mnCost=0;
  while(1){
    fill(dist, dist+nodes, INF); dist[s] = 0;
    fill(pre, pre+nodes, -1); pre[s]=0;
    fill(cap, cap+nodes, 0); cap[s] = INF;
    priority_queue<pair<ll,int> > q; q.push(make_pair(0,s));
    //~ Bellman Ford
    //~ forn(i,nodes) {
      //~ forn(j,sz(e)) if (e[j].rem()) {
        //~ ll c = e[j].cost + pot[e[j].u] - pot[e[j].v];
        //~ if (dist[e[j].v] > dist[e[j].u] + c) {
          //~ dist[e[j].v] = dist[e[j].u] + c;
          //~ pre[e[j].v] = j;
          //~ cap[e[j].v] = min(cap[e[j].u], e[j].rem());
      //~ } }
    //~ }
    //~ Dijkstra
    while (!q.empty()) {
      pair<ll,int> top = q.top(); q.pop();
      int u = top.second;
      ll d = -top.first;
      if (u == t) break;
      if (d > dist[u]) continue;
      forn(i,sz(G[u])) {
        edge E = e[G[u][i]];
        ll c = E.cost + pot[u] - pot[E.v];
        if (E.rem() && dist[E.v] > dist[u] + c) {
          dist[E.v] = dist[u] + c;
          pre[E.v] = G[u][i];
          cap[E.v] = min(cap[u], E.rem());
          q.push(make_pair(-dist[E.v], E.v));
        }
      }
    }
    if (pre[t] == -1) break;
    forn(u,nodes)
      if (dist[u] == INF) pot[u] = INF;
      else pot[u] += dist[u];
    mxFlow +=cap[t];
    mnCost +=cap[t]*pot[t];
    for (int v = t; v != s; v = e[pre[v]].u) {
      e[pre[v]].flow += cap[t];
      e[pre[v]^1].flow -= cap[t];
    }
  }
}
```

9. Template

```
#include <bits/stdc++.h>
using namespace std;
#define dprint(v) cerr << #v"=" << v << endl //;)
#define forr(i,a,b) for(int i=(a); i<(b); i++)
#define forn(i,n) forr(i,0,n)
#define dforn(i,n) for(int i=n-1; i>=0; i--)
#define forall(it,v) for(typeof(v.begin()) it=v.begin();it!=v.end();++it)
#define sz(c) ((int)c.size())
#define zero(v) memset(v, 0, sizeof(v))
#define pb push_back
#define fst first
#define snd second
typedef long long ll;
typedef pair<int,int> ii;

int main() {
  freopen("input.in", "r", stdin);
```

```
    ios::sync_with_stdio(0);
  while(){

  }
  return 0;
}
```

10. Ayudamemoria

Cant. decimales

```
#include <iomanip>
cout << setprecision(2) << fixed;
```

Rellenar con espacios(para justificar)

```
#include <iomanip>
cout << setfill('␣') << setw(3) << 2 << endl;
```

Leer hasta fin de linea

```
#include <sstream>
//hacer cin.ignore() antes de getline()
while(getline(cin, line)){
    istringstream is(line);
    while(is >> X)
      cout << X << "␣";
    cout << endl;
}
```

Aleatorios

```
#define RAND(a, b) (rand()%(b-a+1)+a)
srand(time(NULL));
```

Doubles Comp.

```
const double EPS = 1e-9;
x == y  <=> fabs(x-y) < EPS
x >  y  <=> x > y + EPS
x >= y  <=> x > y - EPS
```

Limites

```
#include <limits>
numeric_limits<T>
  ::max()
  ::min()
  ::epsilon()
```

Muahaha

```
#include <signal.h>
void divzero(int p){
  while(true);}
void segm(int p){
  exit(0);}
//in main
signal(SIGFPE, divzero);
signal(SIGSEGV, segm);
```

Mejorar velocidad

```
ios::sync_with_stdio(false);
```

Mejorar velocidad 2

```
//Solo para enteros positivos
inline void Scanf(int& a){
  char c = 0;
  while(c<33) c = getc(stdin);
  a = 0;
  while(c>33) a = a*10 + c - '0', c = getc(stdin);
}
```

Expandir pila

```
#include <sys/resource.h>
rlimit rl;
getrlimit(RLIMIT_STACK, &rl);
rl.rlim_cur=1024L*1024L*256L;//256mb
setrlimit(RLIMIT_STACK, &rl);
```

C++11

```
g++ --std=c++11
```

Leer del teclado

```
freopen("/dev/tty", "a", stdin);
```

Iterar subconjunto

```
for(int sbm=bm; sbm; sbm=(sbm-1)&bm)
```

File setup

```
//tambien se pueden usar comas: {a, x, m, l}
touch {a..l}.in; tee {a..l}.cpp < template.cpp
```